HELSINKI UNIVERSITY OF TECHNOLOGY
Telecommunications Software and Multimedia Laboratory
Tik-111.590 Research Seminar on Digital Media
Fall 2000: Extensible Markup Language (XML)

XML Parsers and XSL Processors

Jani Utriainen

41836V

Abstract

This seminar paper concentrates on XML (Extensible Markup Language) parsers and XSL (Extensible Stylesheet Language) processors. XML parsers are used for reading XML formatted data, either just to ensure that it is properly formed or to store it in the system memory for further processing. XSL processors are used to transform and format XML data to different presentation formats.

There are two architecturally different major types of XML parsers: SAX (Simple API for XML) and DOM (Document Object Model) parsers. SAX parsers are event-based; they generate events during parsing whereas DOM parsers build tree representations out of the XML data. Parsers can also be categorized according to their validation capabilities. Non-validating parsers are only able to check if an XML document is well-formed whereas validating parsers can additionally check if the XML document conforms to its DTD (Document Type Definition). Currently there are some parser implementations available in all categories although their capabilities and quality varies extensively.

XSL processing is a complex task involving transforming tree structured XML data in several iterations. Most XSL processors are only capable of doing XSLT (XSL Transformations) leaving the formatting part, which is the other half of XSL, out. These processors come in different forms and functionality, which makes choosing a processor difficult. The XSL part of the XML specification is new and still evolving resulting in that only the biggest joint developer groups are able to keep up and implement the latest changes.

XML seems to be ready for serious applications. There are a sufficient number of good enough quality parsers and XSL processors available. Clearly there are a lot of developers and designers who believe in XML and they have put vast amounts of work into different tools, which now are ready to be used by anyone.

Contents

1. Introduction

Sun Microsystems developed Java, which we all know is a programming language that produces highly *portable code*. Now not only Sun, but other parties as well, are delighted about XML, which is to deliver us *portable data*. [22]

XML (Extensible Mark-up Language) is a meta-language used to describe logical structures of documents. With XML, it is possible to give a description of the actual content of each part in the document and describe the roles, which each content section takes in the document. The term "document" really has got a broadened meaning in the XML word. Apart from traditional documents, it also comprises textual files describing graphical objects, transactions, protocol data units and all other kind of imaginable structured data. [24]

XML itself defines only a framework, which as such has a limited value. The value of XML based solutions is based on the technologies and extensions built upon the XML core specification. [12]

The actual mark-up language is based on tags just like SGML and HTML. The tags in XML can be arbitrary unlike in HTML, which uses a strict, although at times expanding set of specified tags. It is not however designed to replace SGML, which is a far more extensive mark-up language. XML was specified by the World Wide Web Consortium (W3C) in the late 90s. The XML specification family consists of several recommendations and drafts. The most important recommendations are the Extensible Markup Language (XML) 1.0 (2nd Edition), the XSL Transformations (XSLT) Version 1.0, the XML Path Language (XPath) Version 1.0 and Namespaces in XML. [5]

When the W3C developed XML, it was to be human-legible, easy to process, support a variety of applications, easy to use over the Internet, compatible with SGML and the specification had to be formal and concise [3]. That is pretty much what XML is today. In addition, XML was developed fairly quickly and developers around the world have already started to adopt it in their designs and systems.

A lot of tools have been developed for creating and manipulating XML based data. This seminar paper concentrates on parsing and validating XML documents and on tools and libraries used for applying stylesheets on existing documents. XSL (Extensible Stylesheet Language) is used for transforming an XML document into an easier viewable form like a book, newspaper, web page, etc. Today, there are a lot of tools for parsing and transforming XML documents; unfortunately these tools are of varying functionality and quality.

It has often been heard that XML is one of those "technologies of the future". So far companies have been cautious about integrating XML into their business critical applications but it seems that XML is now ready for the challenge. XML complements nicely the idea of easy portability that we already got fond of through Java. And now when XML has been extended for yet more applications it seems that the time for utilization is right.

2. XML Parsers

2.1 What Is an XML Parser?

An XML parser is a tool or library, basically a piece of software that checks if the document is syntactically well-formed. Some parsers are also able to check the document's validity by running additional checks on it. [12][24]

The content of a document is always revealed during or after the parsing process. There are two types of parsers, which of event based parsers allow the applications to process the contents during paring whereas tree based parsers only reveal contents after parsing is complete. [12]

These clear differences make it easy to categorize different parsers into four groups (*see figure 1*). A clear categorization also makes it easier to choose a parser for a certain task.

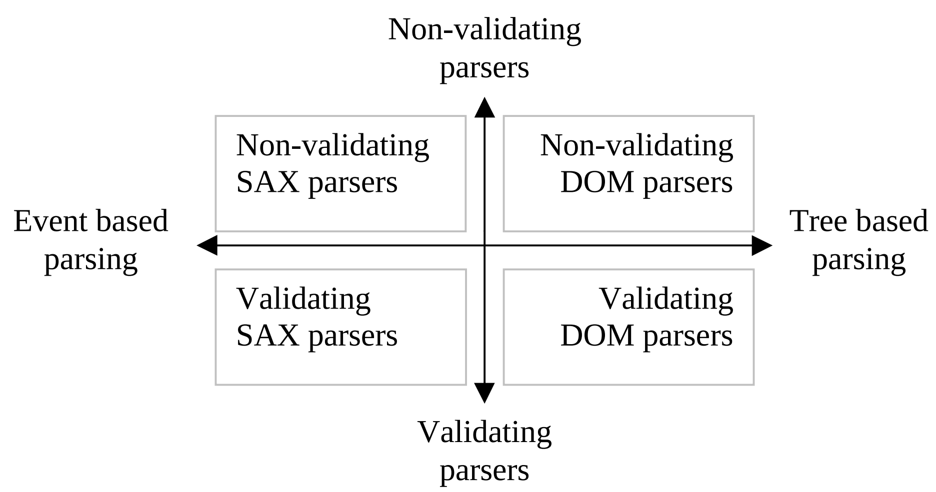

Figure 1. Types of XML parsers.

2.1.1 Classification by Validation

2.1.1.1 Non-validating Parsers

Non-validating XML parsers process all the declarations read from an XML document starting from the document entity (root element). This includes normalizing attribute values, including replacement texts and supplying default values. If the document meets all well-formedness constraints mentioned in the XML specification, it is well-formed. [3]

Simplifying slightly, a well-formed document must conform to the following [24]:

- The document must conform to the XML grammar.
- The replacement text for all parameter entities referenced inside a mark-up declaration must consist of zero or more complete mark-up declarations.
- No attribute may appear more than once in the same tag.
- String attribute values cannot contain references to external entities.
- Non-empty tags must be properly nested.
- Parameter entities must be declared before they are used.
- All entities except the following: `amp`, `lt`, `gt`, `apos`, and `quot` must be declared.
- A binary entity cannot be referenced in the flow of content.
- Neither text nor parameter entities are allowed to be recursive, directly or indirectly.

If a document is well-formed it is an XML document.

2.1.1.2 Validating Parsers

A valid XML document is one that is well-formed (therefore conforming to what is discussed in 2.1.1.1), has a Document Type Declaration (DTD) and conforms to it. A DTD brings out the overall structure and syntax of a document by defining the grammar and vocabulary of the mark-up language used. In short, the DTD specifies everything a parser needs to know in order for that parser to interpret a well-formed XML document. [24][3][21]

In order to check the validity, validating parsers have to do some additional work compared to non-validating parsers. They have to read and process the entire DTD and all external parsed entities referenced in the document and apply this information to constrain the XML document under parsing. [3]

Validating parsers perform slower than non-validating parsers, which is quite understandable considering the extra work they have to do with the DTDs. Therefore when choosing a parser, one should spend some time in analyzing the actual requirements carefully. To be on the sure side it is better to choose a validating parser, since most validation parsers work as plain non-validating parsers too.

2.1.2 Classification by API

There are two major types of XML APIs in use today: event-based and tree-based. An event-based API defines an event interface that applications can use to serve up the document's information items as they are parsed. A tree-based API constructs an object model that represents the XML document's logical structure, which then is made available to the application after the document has been parsed. [20]

2.1.2.1 Simple API for XML (SAX)

The Simple API for XML (SAX) is a simple, low-level event-based API specification [12] [20]. It is not an XML parser itself but an interface that is commonly used when implementing event-based parsers. SAX was developed collaboratively by the members of the XML-DEV mailing list, hosted by OASIS. SAX 2.0 was released on the 5th of May 2000, and is free for both commercial and non-commercial use. [13]

SAX reports parsing events (such as the start and end of elements) directly to the application through callbacks. SAX parsers do not usually build an internal tree like DOM parsers do. The application using a SAX parser implements handlers to deal with the different events, much like handling events in a graphical user interface. With SAX it is possible to parse documents much larger than the available system memory and construct any kind of data structures using callback event handlers. [13]

The fact that SAX parsers consume less memory than tree-based (DOM) parsers and that it is possible to get the desired data without parsing the whole document, are something to consider when choosing a parser implementation.

2.1.2.2 The Document Object Model (DOM)

The Document Object Model (DOM) is the standard tree-based API specification [20]. As with SAX, DOM is not an XML parser itself but only an interface that is commonly used when implementing tree-based parsers. A tree-based parser parses and compiles an XML document into an internal, in-memory tree structure, then allows an application to navigate that tree. DOM is under constant development by the Document Object Model working group at the W3C (World Wide Web Consortium). The current version is DOM Level 2 proposed recommendation, issued on the 27th of September 2000. [11]

The DOM Level 2 Core is a platform- and language-neutral interface that allows applications to dynamically access and update the content and structure of documents. The core also contains specialized interfaces dedicated to XML. [7]

Since a DOM parser always constructs an internal tree representation of the XML document content, it consumes memory proportional to the size of the document. This may become a problem with large documents. However, DOM does allow applications to perform tree operations such as node additions, modifications, conversions and removals. Later on when dealing with XSL processors, these are required operations for XSL processors.

2.2 Publicly Available Parsers

When choosing a parser, there are things that should be considered. The requirements at hands should be analyzed carefully and a parser that suits these requirements best should be chosen. When dealing with XML parsers it seems that most unneeded features increase resources consumption (execution time or memory). The large variety of available parsers gives developers a lot of freedom in making trade-off between consumed system resources and additional functionality.

Checklist of features to consider when choosing a parser:

- Surely it checks for well-formedness but is it validating? Do I need validation?

- Does it have DOM Level 1 support? Do I need that?
- Does it have DOM Level 2 support? Do I need that?
- Does it have SAX version 1 support? Do I need that?
- Does it have SAX version 2 support? Do I need that?
- Does it have XML Schema support? Do I need that?
- Does it support other cool stuff **that I need**?
- Which programming languages does it support? Does it support mine?
- Is it free? How much does it cost?
- Is the source code available?

2.2.1 Free Parsers

Below is a table of parsers that are free of charge.

Name	**From**	**Validity**	**DOM1**	**DOM2**	**SAX1**	**SAX2**	**Other**	**Languages**	**Open Src**
Ælfred[1]	Microstar Software Ltd.							Java	x
Expat	James Clark	x						C	x
Java™ Project X[2]	Sun	x	x		x			Java	x
Lark	Tim Bray	x						Java	
Larval[3]	Tim Bray	x						Java	
LT XML	LTG Software	x						C	x
Microsoft XML parser	Microsoft	x	x	x		x	XML Namespaces	C++, VB, VB script	
OpenXML	Exolab?		x	nf	x	nf		Java	x
PyXML			x	x	x			Python	x
SXP	Silfide	x	nf		x		XML Namespaces, XML links and pointers	Java	x
TclXML[4]	Zveno							Tcl	x
Xerces	Apache	x	x	x		x	XML Schema	Java, C++, Perl	x
XML4J[5]	IBM	x	x	e	x	e	XML Schema	Java	x
XP	James Clark				x			Java	x

x = supports, e = experimental support, nf = not fully supporting, b = supported in beta version

[1] Ælfred is designed for production use in applets.
[2] Java Project X is an experimental, unsupported technology
[3] Based on Lark
[4] Two versions available: a wrapper for Expat and a parser like expat but written in Tcl.
[5] Based on the Apache Xerces XML parser

2.2.2 Commercial parsers

Name	From	Validity	DOM1	DOM2	SAX1	SAX2	Other	Languages	Open Src
Oracle XML Parser	Oracle	x	x	b	x	b	XML Namespaces, XSL transformations	Java, C, C++, PL/SQL	
Resin	Caucho	?	x	x	x	x		Java	
XMLBooster[6]	PhiDaNi Software	x						C, COBOL, Delphi, Java	

x = supports, b = supported in beta version

2.2.3 Online Parsers

Online parsers are able to check for document well-formedness and validity. They are located on the Internet and they are operated with a web browser. There is no need to download any software.

2.2.3.1 Well-formedness Checkers

XML.com's "RUWF? The XML Syntax Checker":
http://www.xml.com/xml/pub/tools/ruwf/check.html

W3C's "HTML Validation Service":
http://validator.w3.org/

IBM's "DOMit: A servlet for XML validation":
http://www.networking.ibm.com/xml/XmlValidatorForm.html

IBM's "Check via XML4J":
http://www.oasis-open.org/cover/xml4j-check00.html

Microsoft's "XML Validator":
http://msdn.microsoft.com/downloads/samples/internet/xml/xml_validator/sample.asp#

2.2.3.2 Validity Checkers

Richard Goerwitz's "STG XML Validation Form":
http://www.stg.brown.edu/service/xmlvalid/

Richard Tobin's "XML well-formedness checker and validator"
http://www.cogsci.ed.ac.uk/~richard/xml-check.html

[6] A meta-parser, which produces source code for parsers in C, COBOL, Delphi and Java as defined. These sources can be compiled for maximum parsing performance.

2.3 Usage Examples

2.3.1 Java

2.3.1.1 Sun's Java™ API For XML Processing (JAXP)

The following is an abstract example of how to configure a DOM parser to be namespace aware and validating. Notice, the API for XML Processing is only a standardization draft and there is no implementation of it at current time [6]:

```
DocumentBuilder builder;
DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
factory.setValidating(true);

try {
      builder = factory.newDocumentBuilder();
      Document document =
            builder.parse("http://myserver/mycontent.xml");
} catch (SAXException se) {
      // handle error
} catch (IOException ioe) {
      // handle error
} catch (ParserConfigurationException pce) {
      // handle error
}
```

2.3.1.2 Apache Xerces

The following is an example of how to configure a SAX parser to be namespace aware and validating [2]:

```
Parser parser =
      ParserFactory.makeParser("org.apache.xerces.parsers.SAXParser”);

try {
      parser.setFeature("http://xml.org/sax/features/validation", true);
      parser.parse("http://myserver/mycontent.xml");
} catch (SAXException se) {
      // handle error
} catch (IOException ioe) {
      // handle error
}
Document document = parser.getDocument();
```

When using SAX2.0, Xerces is namespace aware by default. Sometimes this causes annoyance when validating documents against some DTDs. This is due to the fact that XML 1.0 and DTDs cannot distinguish between prefixes and element names. To turn off namespace awareness add the following line:

```
parser.setFeature("http://xml.org/sax/features/namespaces", false);
```

2.3.2 Visual Basic And MSXML

The following example loads (and parses) an XML document into a DOM tree with MSXML 2.5 [17]:

```
Dim xmlDoc
Set xmlDoc = CreateObject("Microsoft.xmldom")
xmlDoc.async = False
xmlDoc.load("c:\books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
       MsgBox ("A parse error occurred.")
Else
       MsgBox xmlDoc.documentElement.xml
End If
```

The following example uses SAX to parse an XML document with MSXML3.0 [16]:

```
Dim reader As New SAXXMLReader
On Error GoTo 10
reader.parseURL ("http://myserver/mycontent.xml");
Exit Sub

10: MsgBox("*** Error *** " & Err.Number & " : " & Err.Description)
End Sub
```

3. XSL Processors

3.1 XML Transformations And Formatting

XML transformations and formatting are done in order to filter and convert XML data into different forms for various clients. These clients can be human end users or client applications. Common to these clients is their need of their own representation of the underlying data. Humans want to read their information (which is actually just embellished data) in a human-readable, textual form whereas computer applications may want their input in some simpler form, which may not even be XML!

From a presentation point of view, the advantages of separating the contents and style is being able to reuse content data. The same content should look different in different contexts. By separating content and style, it is possible to publish the same content in different media and devices like newspapers, books, pamphlets, PC web browsers, handheld devices, electronic books, banners, etc. The presentation style can also adapt to reader disabilities by for example adjusting text sizes and colors. [9]

Although transformations and formatting seem useful they are not easy to implement. Several separate recommendations, which define how XML processing is done have come out. The most important three (XSLT, XSL-FO and XPath), which are tied tightly together, are commonly grouped under the terms XSL and XSL Transformations. The loose use of these names is not technically correct but widely used in practice. The terms XSL and XSLT are also used to describe the whole transformation process in overall. [12]

3.1.1 XSL

XSL (Extensible Stylesheet Language) is defined as a language for expressing stylesheets. It builds on prior work on Cascading Style Sheets and the Document Style Semantics and Specification Language [1]. XSL itself is well-formed, valid XML. It actually consists of two languages, not only one: a transformation language for XML documents and an XML vocabulary for specifying the formatting of XML documents [12][1][18][10].

The transformation language, which deals with moving data from one document to another, can be used independently of the formatting language, which is more focused on actual presentation issues [12].

Style definition files written in XSL are called stylesheets and usually ends with a “.xsl” extension.

3.1.2 XSL Transformations (XSLT)

XSLT (XSL Transformations) is a language that specifies the conversion of a document from one textual format to another. Although XSLT is designed primarily for XML-to-XML transformations, it is possible to output any other textual formats as well, e.g.

HTML (Hypertext Mark-up Language) and WML (Wireless Mark-up Language). In order to achieve binary output formatting objects must be used. [12] [10]

In an XSL transformation, both the XML document and the XSL stylesheet is read. Processing is done according to the instructions in the XSL stylesheet and this generates new, usually tagged output. XSLT provides mechanisms for advanced pattern matching, inserting and copying data and document fragments, use of conditionals and accessing data in an absolute and relative path structure. XPath is usually heavily used with XSLT.

3.1.3 XSL Formatting Objects (XSL-FO)

XSL Formatting Objects (XSL-FO) is the second half of XSL. It is based on a large model, called formatting model, which defines a set of more than fifty objects that can be applied on an XML document [12]. With XSL-FO it is possible to create a complete description (a stylesheet) of how pages should look like when presented to a client or reader. Generally, a stylesheet uses XSLT to transform an XML source document into a XML result document and a then XSL-FO to produce a suitable presentation format.

Formatting objects are heavily used when converting XML documents into binary formats, such as PDF [12].

3.1.4 XML Path Language

XPath (XML Path Language) is not actually a part of XSL, it is just used in XSL transformations as well as in other specifications published by the W3C. It provides a mechanism for referencing specific parts in an XML document, such as elements, attributes, text fragments, etc. [9][10][4] Sometimes, due to the complexity and size that some XML documents have, locating specific data objects is a difficult task. The actual referencing is done with relative paths. These paths can be relative to the current element or the root element in similar to what we all are accustomed to with operating system directory structures.

XPath defines compact, non-XML syntax for pattern matching. It also provides facilities for manipulating strings, numbers and booleans. In addition to these, XPath also defines some basic functions, such as not() and count(). When using XPath for pattern matching, it is possible that an XPath expression results in no data, a single element or attribute or several elements and attributes, that is, ranges of data. These characteristics make XPath very useful in XSL transformations. [4]

3.2 What Is an XSL Processor?

An XSL processor reads both an XML document and an XSL stylesheet and produces the presentation of that XML content as intended by the designer of that stylesheet [1]. XSL processors must be able to transform XML input into any format as specified in the XSL stylesheet [19].

3.2.1 Tree Representations

Structured data, such as XML, can easily be represented as a tree model. A tree model is fairly simple to traverse and alter, which makes it an ideal model when doing heavy processing on XML data. The negative impact is high memory consumption, since the whole tree has to be parsed into the system memory before processing. The larger amount of data, the bigger the tree grows and the more strain is caused on our system resources. However, the benefits of using tree structures are greater than substantial use of memory resources. Thus all transformations are done in trees.

3.2.2 XSL Stylesheets

An XSL stylesheet is a well-formed, valid XML document basically consisting of a set of templates. An XSL template is defined as a set of rules that should be applied to XML that matches a specific XPath expression. Each template matches some set of elements in the source tree (pattern matching) and then describes the contribution that the matched element makes to the result tree. Stylesheets also contain control structures and other specialized element types from namespaces mentioned in the XSL recommendation. [12][9]

Without an XSL stylesheet there cannot be any XSL processing.

3.2.3 Transforming And Formatting

Before XSL processing can begin, both the XML document and the XSL stylesheet has to be read. Most XSL processors read the XML document into a source tree with one of the parsers mentioned in chapter 2. The actual XSL processing is broken into two parts: first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation [1]. Consult figure X for a view of the complete process.

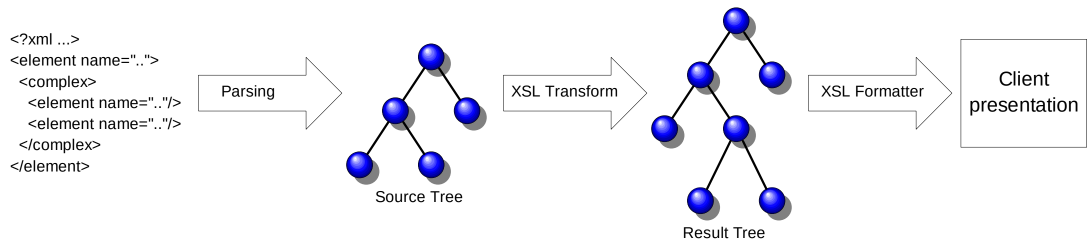

Figure 2. The XML processing with XSL.

Applying the transformation part (basically the XSLT part) of the stylesheet onto the source tree creates a result tree. The tree transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, one could add a table-of-contents as a filtered selection of an original source document, or one could rearrange source data into a sorted tabular presentation. [1]

Interpreting or formatting the result tree produces the presentation of the XML data as intended by the designer of the stylesheet. The result tree is converted (with XSLT) to a formatting object tree. Each node in this tree is a formatting object representing styling information that will be applied to the content of that formatting object. The XSL Formatting Objects specification defines a vocabulary that is used by the designer to specify formatting for each or chosen formatting objects. The formatting object tree is refined in an iterative fashion. Finally the refined formatting object tree is generated into an area tree, which defines how objects are displayed: their position, colors, paddings, borders, etc. Reading the area tree reveals the final presentation. [1]

3.3 Publicly Available XSL Processors

As like the case with parsers, there is a mixed assortment of XML processors available. They come in all different shapes, bundled with optional features and varying quality. It is utmost important to recognize your own needs and choose a processor that is capable of performing those identified tasks.

Checklist of features to consider when choosing a processor:

- Does it have XSLT support?
- Does it have XPath support?
- Does it have formatting objects support?
- Does it have any other features **that I need**?
- Which programming languages does it support? Does it support mine?
- Is it free? How much does it cost?
- Is the source code available?

3.3.1 Free Processors

Below is a table of XML processors that are free of charge.

Name	From	XSLT	XPath	XML-FO	Input	Other	Languages	Open Src
Cocoon	Apache	x	x	x		Full suite	Java	x
FOP	Apache			x	Xalan, DOM, SAX		Java	x
Microsoft XSL processor	Microsoft	?	?	?	DOM		C++, VB, VB script	
Sablotron	Ginger Alliance	x	x		Expat		C++	x
SAXON	Michael Kay	x	x		Improved Ælfred, SAX	Initial TraX support	Java	x
TransforMiiX	Mozilla	x	?		Expat		C++	x
Xalan	Apache	x	x		Xerxes, DOM, SAX	TRaX	Java, C++	x
XT	James Clark	x	?		DOM, SAX		Java	x
XSL:P	Keith Visco	x	?		XML4J, DOM		Java	x

x = supports, ? = don't know!

3.3.2 Commercial Processors

Name	From	XSLT	XPath	XML-FO	Input	Other	Languages	Open Src
iXSLT	Infoteria	x	?		?		C++, ASP, VB	
LotusXSL	IBM	x	x		XML4J, DOM, SAX		Java	
Oracle XML Parser	Oracle	x	?		Oracle XML Parser		Java, C, C++, PL/SQL	
Resin	Caucho	x	x		Resin Parser		Java	

x = supports, ? = don't know!

3.4 Usage

3.4.1 Apache Xalan (Java)

The following is an example of how simple it is to perform an XSL transformation with Apache's Xalan [23]:

```
try {
      // Have the XSLTProcessorFactory obtain a interface to a
      // new XSLTProcessor object.
      XSLTProcessor processor = XSLTProcessorFactory.getProcessor();

      // Have the XSLTProcessor processor object transform "foo.xml" to
      // System.out, using the XSLT instructions found in "foo.xsl".
      processor.process(new XSLTInputSource("foo.xml"),
                        new XSLTInputSource("foo.xsl"),
                        new XSLTResultTarget(System.out));
} catch (java.io.IOException ex) {
      // handle error
} catch (java.net.MalformedURLException ex) {
      // handle error
} catch (org.xml.sax.SAXException ex) {
      // handle error
}
```

3.4.2 Microsoft IE5.0 Client Side Browser XSL Processing

The following is an example of how an XSL transformation can be done in Microsoft Internet Explorer 5.0 or higher. A sample web page is displayed below [14]:

```
<HTML>
  <HEAD>
    <TITLE>Simple demo of Microsoft XSL Processor</TITLE>
  </HEAD>

  <XML id="source" src="simple.xml"></XML>
```

```
  <XML id="style" src="simple.xsl"></XML>

  <SCRIPT FOR="window" EVENT="onload">
    xslTarget.innerHTML = source.transformNode(style.XMLDocument);
  </SCRIPT>

  <BODY>
    <P>
      <B>This demo shows the use of data islands for loading XML source and
       XSL style sheets and inserting the transformed result into the Web page.
     </B>
    </P>
    <DIV id="xslTarget"></DIV>
  </BODY>
</HTML>
```

3.4.3 ASP Server Side XSL Processing

The following is an example of how an XSL transformation can be done with Active Server Pages [15]:

```
<%@ LANGUAGE = JScript %>
<%
  // Set the source and style sheet locations here
  var sourceFile = Server.MapPath("simple.xml");
  var styleFile = Server.MapPath("simple.xsl");

  // Load the XML
  var source = Server.CreateObject("Microsoft.XMLDOM");
  source.async = false;
  source.load(sourceFile);
  // Load the XSL
  var style = Server.CreateObject("Microsoft.XMLDOM");
  style.async = false;
  style.load(styleFile);
  Response.Write(source.transformNode(style));
%>
```

4. Conclusions

XML is still young and it reflects on the available tools. The XML recommendation has recently been updated and the XSLT and XPath specifications have just reached recommendation level. There are a lot of XML related drafts around. This gives little assurance on how the future will be and it raises carefulness amongst XML tool implementers. It also seems that only the big joint groups have enough resources to follow up with the changes and new specifications that constantly are developed. The good news is that the most important specifications are now quite stable and they form a solid ground to build tools on.

Although the field is young, XML parsers and processors have been evolving for some time now. There is a lot of effort that already has been put into generalizing tools and interfaces. Especially Java parsers and processors have got a tremendous push from standardization efforts contributed by Sun Microsystems. Especially in the Java programming development section, there are a lot of alternatives on which parsers and processors to use. Although most XSL processors still lack proper formatting capabilities, they are sophisticated enough for all-around text-based transformations. The time for large-scale utilization is now here and it does not cost that much to evaluate the technology. XML does have its own share to give in large, distributed applications.

References

[1] Adler, Sharon & Berglund, Anders & Caruso, Jeff & Deach, Stephen & Grosso, Paul & Gutentag, Eduardo & Milowski, Alex & Parnell, Scott & Richman, Jeremy & Zilles, Steve, Extensible Stylesheet Language (XSL), 21 November 2000, http://www.w3.org/TR/xsl/

[2] Apache Software Foundation, Xerces Java Parser Readme, http://xml.apache.org/xerces-j/index.html

[3] Bray, Tim & Paoli, Jean & Sperberg-McQueen, C.M. & Maler, Eve, Extensible Markup Language (XML) 1.0 (Second Edition), W3C, 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[4] Clark, James & DeRose, Steve, XML Path Language (XPath) Version 1.0, W3C, 16 November 1999, http://www.w3.org/TR/xpath.html

[5] Conolly, Dan, Extensible Markup Language (XML), W3C, April 1997, http://www.w3.org/XML/

[6] Davidson, James Duncan & Mordani, Rajiv, Java API for XML Processing, Version 1.1 Public Review, http://java.sun.com/aboutJava/communityprocess/review/jsr063/jaxp-pd1.pdf

[7] Davis, Mark & Le Hors, Arnaud & Le Hégaret, Philippe & Robie, Jonathan & Wood, Lauren, Document Object Model (DOM) Level 2 Core Specification, http://www.w3.org/TR/DOM-Level-2-Core

[8] DuCharme, Bob, HTML and XSLT, O'Reilly, 30th August 2000, http://www.xml.com/pub/2000/08/30/xsltandhtml/index.html

[9] Grosso, Paul & Walsh, Norman, XSL Concepts And Practical Use, 12 June 2000, http://www.nwalsh.com/docs/tutorials/xsl/xsl/slides.html

[10] Harold, Elliotte Rusty, The XML Bible, IDG Books Worldwide, USA, 1999, 1015 p.

[11] Lé Hegaret, Philippe, Document Object Model (DOM), W3C, 20th November 2000, http://www.w3.org/DOM/

[12] McLaughlin, Brett, Java™ and XML, O'Reilly, USA, 2000, 479 p.

[13] Megginson, David, SAX 2.0: The Simple API for XML, http://www.megginson.com/SAX/

[14] Microsoft Developers Network, Transforming the Contents of XML Data Islands Using XSL, 1999,
http://msdn.microsoft.com/library/psdk/xmlsdk/xslp3964.htm

[15] Microsoft Developers Network, Using XSL in an Active Server Page, 1999,
http://msdn.microsoft.com/library/psdk/xmlsdk/xslp0e5s.htm

[16] Microsoft, JumpStart for Creating a SAX2 Application with Visual Basic,
http://msdn.microsoft.com/xml/articles/vbsax2jumpstart.asp

[17] Microsoft, XML Developer's Guide,
http://msdn.microsoft.com/library/default.asp?PP=/library/toc/psdk/psdk4-2-2.xml&tocPath=psdk4-2-2&URL=/library/psdk/xmlsdk/xmlp91b9.htm

[18] Microsoft, XSL Reference, 1999,
http://msdn.microsoft.com/library/psdk/xmlsdk/xslr8ko5.htm

[19] Shantaram, Mahesh, Time To Serve XML, Cyber India Online Limited, 30 November 1999,
http://216.46.224.165/content/technology/techbytes/99113001.asp

[20] Skonnard, Aaron, The XML Files: XPath, XSLT, and other XML Specifications, MSDN Magazine, May 2000,
http://msdn.microsoft.com/msdnmag/issues/0500/xml/xml0500.asp

[21] Sol, Selena, WDVL: Introduction to XML For Web Developers, internet.com, 1st March 1999,
http://wdvl.internet.com/Authoring/Languages/XML/Tutorials/Intro/

[22] Sun Microsystems, Sun Unveils XML Technology Support In The Java Platform, 99/03/09, Palo, Alto,
http://www.sun.com/smi/Press/sunflash/9903/sunflash.990309.3.html

[23] The Apache Software Foundation, Sample code from the Xalan version 1.2.1 distribution (SimpleTransform.java)

[24] Walsh, Norman, A Technical Introduction to XML, O'Reilly, 3rd October 1998,
http://www.xml.com/pub/98/10/guide0.html