XML Primer

Contents

1. Introduction

1.1 History

In 1969, Charles Goldfarb of IBM with Edward Mosher and Raymond Lorie defined the Generalized Markup Language: GML (or Goldfarb, Mosher and Lorrie). This was one of the first systems that provided generic markup; it allowed you to define your own markup terms and had a validating parser (the user could say what markup was allowed where). By 1973, IBM had an implementation of GML as part of its Advanced Text Management System.

In 1978, Goldfarb led a project to produce an American standard text description language based on GML. This became SGML (Standard Generalized Markup Language) and was moved to ISO to become an international standard. It finally became an ISO standard in 1986.

SGML provided all the mechanisms to define a markup language for a particular purpose. It did not constrain the tags that could be used to markup the text nor did it constrain what symbols would be used for the opening start-tag, opening end-tag, closing start-tag and closing end-tag delimiters.

SGML also defined a **Document Type Definition** which accompanies the document and defines the structure of tags that are allowed. ISO defined a separate standard called **DSSL**, the Document Style Semantics and Specification Language which described the presentation formatting required by the document's author.

SGML was widely used at CERN for documentation starting as early as 1984. CERN documents were defined in the CERN SGML GUID language. When Tim Berners-Lee defined HTML for the Web, he had been an SGML GUID user for a number of years.

The current move is away from HTML and towards application-specific markup using XML (Extensible Markup Language). XML derives from SGML. This means that a reformulation of HTML as an application of XML makes a great deal of sense. The facilities available in SGML and the tools surrounding SGML become available.

XML is a subset of SGML with the goal of allowing SGML documents to be served, received, and processed on the Web in the way that is now possible with HTML. XML was initially developed by the W3C SGML Working Group chaired by Jon Bosak of Sun Microsystems. XML is primarily aimed at the requirements of **large-scale Web content providers for industry-specific markup, vendor-neutral data exchange, media-independent publishing, one-on-one marketing, workflow management in collaborative authoring environments, and the processing of Web documents by intelligent clients**. Valid XML documents were designed to be valid SGML documents also.

The main participants in the devlopment of XML early on were Jon Bosak, Tim Bray, James Clark, Dan Connolly, Steve DeRose, Dave Hollander, Eliot Kimber, Tom Magliery, Eve Maler, Jean Paoli, Peter Sharpe and C. Michael Sperberg-McQueen. Several of these are still active in XML developments.

1.2 What is Wrong with HTML

SGML allowed documents to describe their own grammar, the elements, their attributes and the structural relationships that exist between these. On the other hand, HTML has a small set of defined elements. This makes life easy but at the cost of limiting HTML in terms of:

- **Extensibility**: Users cannot define their own elements or attributes
- **Deep Structure**: Highly structured documents can only be produced by embedding lists inside lists inside lists or the same with tables
- **Validity**: HTML is sufficiently lax that it is difficult to define an invalid or valid document

So three major features of XML are:

- Users can define new elements and attribute names as required
- Document structures can be nested to any depth or complexity
- An XML document may contain a description of its grammar so that applications can perform structural validation

Some applications requiring these facilities are:

- Documents which are not simple text documents
- Extraction of information from databases
- Client-side processing
- Applications requiring different presentations of the same data

So the answer to the question is that there is nothing wrong with HTML for the task that it is addressing. That is the production by a large community of mainly textual documents for viewing on the Web. It comes unstuck when it goes outside that domain. For example, I wish to send a student's answers to an exam paper to a colleague for marking and I want to do this via the Web. My markup might be:

```
<html>
<head>
<title> Fred Smith's P08770 Paper </title>
</head>
<body>
<h1>Who invented the Web? </h1>
<p>Tim Berners-Lee</p>
<h1>What year was the Web invented? </h1>
<p>1989</p>
</body>
</html>
```

There are no elements defined for marking up **exam papers**, a **question and answer** pair, a **question** or an **answer**. I have to use the HTML markup for a purpose that it was not intended. I choose h2 as the markup for question and p as the markup for answer. I need to tell my colleague what the markup translation is. There are no constraints imposed by HTML in ensuring that every question has an answer and that a question does not have two answers. Such constraints are beyond HTML.

1.3 Why not SGML

The major problem with SGML is age. It was developed in an era of punched cards and the need to keep keystroke input to a minimum. In consequence it suffers some of the lack of clarity that exists with HTML.

To a large extent, XML is what SGML would have been if it had been defined when XML was needed. XML is a subset of SGML that retains all the major features and removes some of the laxness of SGML and some of the richer features primarily defined for the large scale documentation area.

1.4 XML History

The first Working Draft of XML appeared in November 1996 but attracted less attention than the new Java language that was gaining all the headlines at the time. Java had been announced in the Spring of 1995 and by the Autumn of 1995 it was the hot topic at WWW4 in Boston.

XML 1.0 was first announced to a wide audience at the WWW6 Conference in Santa Clara in April 1997. Incidentally, this was also the launch of the Accessibility Initiative within W3C. Accessibility and XML are of a similar age. Appendix C gives Jon Bosak's paper that was presented at the WWW6 Workshop on XML and gives a good overview of the thinking at that time.

The Working Draft presented at Santa Clara became a W3C standard in February 1998, under a year from the original presentation to a wide audience.

The second significant event was the publication of the XML Namespaces standard in January 1999.

1.5 A Family of Standards

XML is not a single standard. The core XML 1.0 syntactic defintion is enhanced by the following standards that are either complete or under development:

Recommendations

XML 1.0: February 1998 (Revised October 2000)
XML Namespaces: January 1999

XSLT: November 1999 (part of XSL)
XPath: November 1999 (used by XSLT and XPointer)
XHTML (HTML redefined as XML): January 2000
XHTML Basic: December 2000 (Base subset)
Canonical XML: March 2001
XML Schemas Parts 1 and 2: May 2001

XLink and XBase: June 2001
XSL: October 2001
XML Information Set: October 2001
XML-Signature: February 2002

Proposed Recommendations

Exclusive XML Canonicalization: May 2002

Candidate Recommendations

XML Fragment Interchange: February 2001
XPointer: September 2001
XInclude: February 2002
XML Signature Decryption: March 2002
XML Encryption: March 2002

Working Drafts

XML Events: October 2001
XML Protocol: December 2001
XForms 1.0: January 2002
XML Key Management: March 2002
XKMS: March 2002
XPath 2.0: April 2002
XQuery: Arpil 2002
XML 1.1: April 2002

2. XML 1.0

2.1 Role of XML

Early on W3C produced a document giving the seven main reasons for XML:

1. XML is a method for putting structured data in a text file

For "structured data" think of such things as spreadsheets, address books, configuration parameters, financial transactions, technical drawings, etc. Programs that produce such data often also store it on disk, for which they can use either a binary format or a text format. The latter allows you, if necessary, to look at the data without the program that produced it. XML is a set of rules, guidelines, conventions, whatever you want to call them, for designing text formats for such data, in a way that produces files that are easy to generate and read (by a computer), that are unambiguous, and that avoid common pitfalls, such as lack of extensibility, lack of support for internationalization/localization, and platform-dependency.

2. XML looks a bit like HTML but isn't HTML

Like HTML, XML makes use of **tags** (words bracketed by '<' and '>') and **attributes** (of the form `name="value"`), but while HTML specifies what each tag & attribute means (and often how the text between them will look in a browser), XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. In other words, if you see "<p>" in an XML file, don't assume it is a paragraph. Depending on the context, it may be a price, a parameter, a person, a p... (b.t.w., who says it has to be a word with a "p"?)

3. XML is text, but isn't meant to be read

XML files are text files, as I said above, but even less than HTML are they meant to be read by humans. They are text files, because that allows experts (such as programmers) to more easily **debug** applications, and in emergencies, they can use a simple text editor to fix a broken XML file. But the rules for XML files are much stricter than for HTML. A forgotten tag, or an attribute without quotes makes the file unusable, while in HTML such practice is often explicitly allowed, or at least tolerated. It is written in the official XML specification: applications are not **allowed** to try to second-guess the creator of a broken XML file; if the file is broken, an application has to stop right there and issue an error.

4. XML is a family of technologies

There is XML 1.0, the specification that defines what "tags" and "attributes" are, but around XML 1.0, there is a growing set of optional modules that provide sets of tags & attributes, or guidelines for specific tasks. There is, e.g.,Xlink (still in development as of November 1999), which describes a standard way to add hyperlinks to an XML file. XPointer & XFragments (also still being developed) are syntaxes for pointing to parts of an XML document. (An XPointer is a bit like a URL, but instead of pointing to documents on the Web, it points to pieces of data inside an XML file.) CSS, the style sheet language, is applicable to XML as it is to HTML. XSL (autumn 1999) is the advanced language for expressing style sheets. It is based on XSLT, a transformation language that is often useful outside XSL as well, for rearranging, adding or deleting tags & attributes. The DOM is a standard set of function calls for manipulating XML (and HTML) files from a programming language. XML Namespaces is a specification that describes how you can associate a URL with every single tag and attribute in an XML document. What that URL is used for is up to the application that reads the URL, though. RDF, W3C's standard for metadata, uses it to link every piece of metadata to a file defining the type of that data.) XML Schemas 1 and 2 help developers to precisely define their own XML-based formats. There are several more modules and tools available or under development. Keep an eye on W3C's technical reports page.

5. XML is verbose, but that is not a problem

Since XML is a text format, and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the XML developers. The advantages of a text format are evident (see 3 above), and the disadvantages can usually be compensated at a different level. Disk space isn't as expensive anymore as it used to be, and programs like zip and gzip can compress files very well and very fast. Those programs are available for nearly all platforms (and are usually free). In addition, communication protocols such as modem protocols and HTTP/1.1 (the core protocol of the Web) can compress data on the fly, thus saving bandwidth as effectively as a binary format.

6. XML is new, but not that new

Development of XML started in 1996 and it is a W3C standard since February 1998, which may make you suspect that this is rather immature technology. But in fact the technology isn't very new. Before XML there was SGML, developed in the early '80s, an ISO standard since 1986, and widely used for large documentation projects. And of course HTML, whose development started in 1990. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, but vastly more regular and simpler to use. Some evolutions, however, are hard to distinguish from revolutions... And it must be said that while SGML is mostly used for technical documentation and much less for other kinds of data, with XML it is exactly the opposite.

7. XML is license-free, platform-independent and well-supported

By choosing XML as the basis for some project, you buy into a large and growing community of tools (one of which may already do what you need!) and engineers experienced in the technology. Opting for XML is a bit like choosing SQL for databases: you still have to build your own database and your own programs/procedures that manipulate it, but there are many tools available and many people that can help you. And since XML, as a W3C technology, is license-free, you can build your own software around it without paying anybody anything. The large and growing support means that you are also not tied to a single vendor.
XML is not always the best solution, but it is always worth considering.

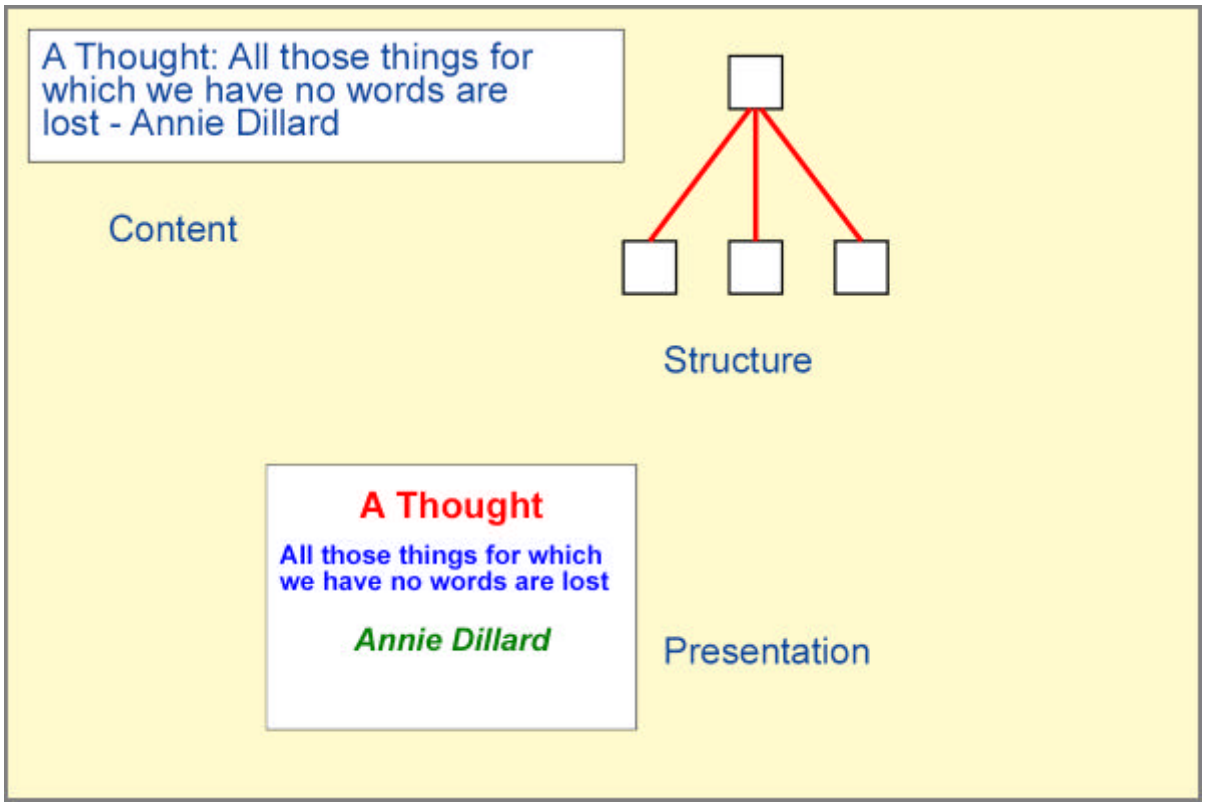

Figure 2.1: Document Structure

2.2 Tags, Elements and Attributes

Figure 2.1 shows a very simple document and a diagram indicating how it could be marked up to indicate the three major parts of the document that define its structure of **title**, **thought** and **author**. Mark-up shows off that strucure. When the document is presented, the structure will normally be shown by the use oof different style as is shown in the Figure.

Figure 2.2 shows the role of XML. Its function is just concerned with marking up the document and indicating its structure. The function of presenting the structure is delegated to either Cascading Style Sheets (CSS) or the Extensible Styling Language (XSL).

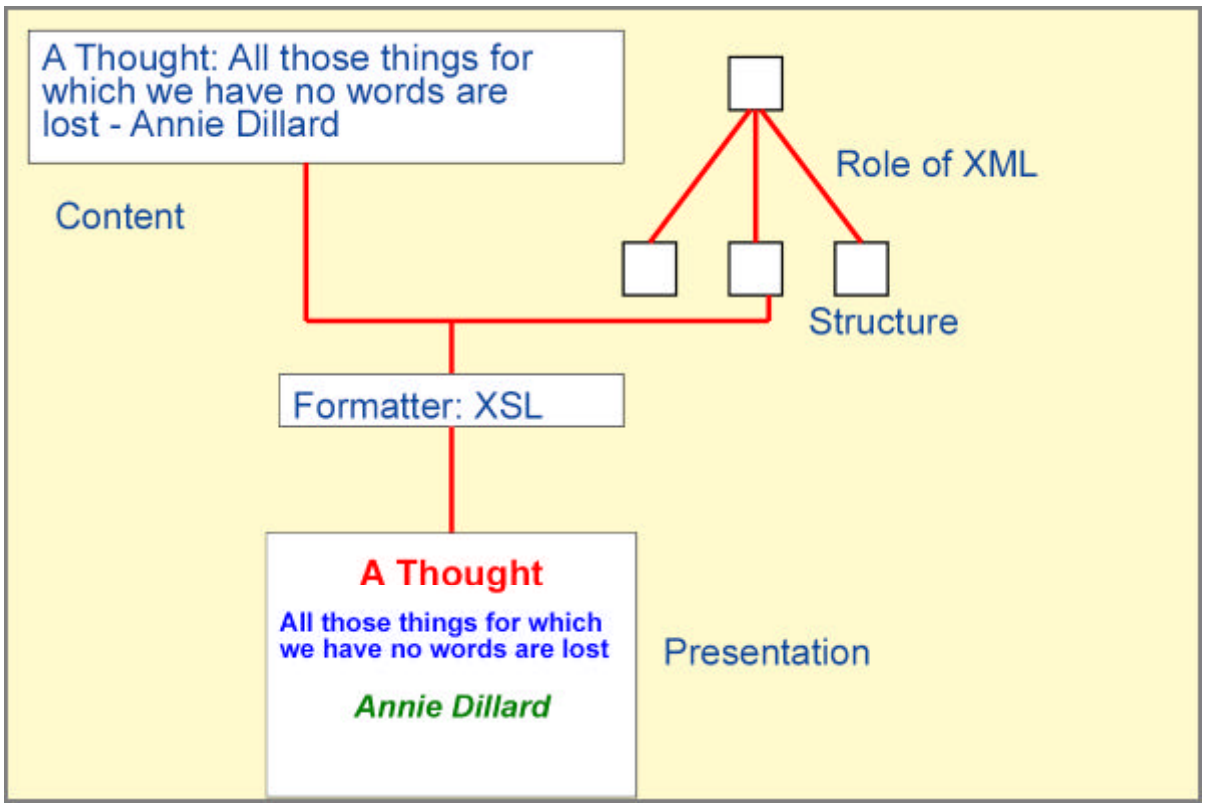

Figure 2.2: Styling a Doxument

XML marks up information using a **start-tag** and an **end-tag** to delimit the bounds of an **element**. A start-tag consists of:

- Left angle bracket (<)
- The tag name (title, say)
- It may have one or more attributes (id="3")
- Right angle bracket (>)

An example of a start-tag is:

```
<title>
```

An end-tag consists of:

- Left angle bracket (<)
- A slash (/)
- The tag name (title)
- Right angle bracket (>)

An example of an end-tag is:

```
</title>
```

An element consists of:

- A start-tag
- The element's content which consists of text and other elements
- The matching end-tag
- Element names are case sensitive in XML

Some example elements are:

```
<title>The title of your document</title>

<Author><surname>Hopgood</surname><firstname>Bob</firstname></Author>

<PARA>This is a <emphasis>large</emphasis> book</PARA>

<rule></rule>

<ramble>This
has additional    spacing
   and newlines scattered
in the text</ramble>
```

With each element, it is possible to associate **attributes**. Attributes are used to qualify the meaning of an element. They appear within the start-tag. The form of an attribute is **name** followed by **=** followed by **value** in quotes. Both double quotes (") and single quotes (') may be used but they must be a pair. For example:

```
<title name="CBE'>
```

is not allowed. Attributes are separated by spaces:

```
<title name="CBE" list="civil">
```

Attributes can only appear once. The following is not allowed:

```
<title name="CBE" name="civil">
```

Looking back at our original thought, a possible markup that brings out its structure would be:

```
<?xml version="1.0"?>
<thought>
  <title>A Thought</title>
  <body>
    All those things for which
    we have no words are lost
  </body>
  <author>Annie Dillard</author>
</thought>
```

The elements **thought**, **title**, **body** and **author** have been defined. If it appears, the XML declaration must start an XML document but it is optional. The characters xml must be lower case. A document contains a single tag pair that defines the **root** element. In this example the root element is the **thought** element

All tag and attribute names must start with an alphabetic character, an underscore (_) or acolon (:). The name must not contain any **whitespace**, that is characters like space, tab, line feed, carriage return etc. Punctuation characters are allowed in names. The colon character is used just to define the **namespace** (which will be discussed later). Names cannot start with **xml**, **xML**, **xmL**, **XmL**, or any other combination.Finally, names in XML are case sensitive. So <exam>, <ExaM>, <EXAM> are all different tags.

2.3 Well-Formed Documents

A **well-formed** XML document:

- Has a single root element
- All other elements are correctly nested

If the XML document is **not** well formed, programs using it must report an error. Well-formed documents can be parsed by a client application without any external resources being required.

Documents may also be defined as **valid** which means that they:

- Abide by the constraints placed on each element's position in the document
- Abide by the constaints placed on the attributes of each element
- Require a **Document Type Definition** or **XML Schema** to specify the constraints

There is good support for XML in IE6.0. It contains both an unvalidating and validating parser (validity defined by a DTD). Any file opened by IE6 of type xxx.xml will have the IE6.0 parser check that it is well-formed and, if it succeeds IE will display a tree structure for the document. If it is not well formed, IE will give an error message.

The main advantage that come with an XML application is that elements can be structured in a way to suit the application. It is no longer necessary to try and fit the document to the elements available in HTML.

2.4 XML Structure

An XML document consists of a **Prologue** followed by the **root element**. There are also some **miscellaneous** information that can appear practically anywhere. The two main parts of the Prologue are:

- XML declaration: **<?xml version="1.0"?>**
- Document Type Declaration: **<!DOCTYPE . . .>**

Miscellaneous information can appear between the two and after the second. Miscellaneous information consists of any of these three:

- Comment
- Processing Instruction
- White Space characters (space, carriage return, line feed, tab)

The root element can, of course, include other elements as part of its content to any depth or complexity that the application requires.

Processing Instructions (PIs)

Processing Instructions are not part of the character data of the XML document itself. They each contain an instruction to the parser. The XML Declaration is a special kind of PI that defines the version of XML to be parsed against. The general form of a PI is:

```
<?ApplicationName InstructionsForTheApplication ?>
```

Comments

Comments in XML are surrounded by <!-- and -->. Two hypens together are not allowed in the comment itself. For example:

```
<!--This is a comment-->
<!--This is -- not a -- comment-->
```

Contents of an XML Element

An XML element can consist of just the **empty-element tag** or a **start-tag** followed by **content** followed by **end-tag**. The content can be a mixture of:

- **character data**
- **element**
- Processing Instruction
- Comment
- **CDATA Section**
- **Reference**

Character data are any characters other than < and &. A **CDATA Section** is a way of enclosing character data so that it can include characters like < and &. To allow special characters such as these two to appear outside a CDATA Section, XML has the concept of a **reference** (entity and character) which starts with an **&** character (which is why that character is not allowed in character data). These will be discussed later.

CDATA Section

Sometimes there is a need to pass a string of characters to an application without parsing them. XML has the reserved characters < and & for example. The following markup is therefore illegal:

```
<answer>27 when x < 3 </answer>
```

The correct way is to put the string not to be parsed in a CDATA section:

```
<answer><![CDATA[27 when x < 3]]> </answer>
```

CDATA Sections are mainly used when the text contains a large and unknown number of reserved characters. For example, a binary encoded image might contain any number of them and they would be difficult to find and would probably invalidate the image even if they were substituted. The only string not allowed in a CDATA section is]]>.

2.5 References

Entity References

Entity references are a way of forcing the parser to replace the **entity reference** with some other data. It allows reserved or special characters to be used in a document. Entity references start with an & and terminate with a ;. Two symbols are reserved and **must** be escaped when used elsewhere: & and <. The symbol > is escaped also for compatibility with SGML although it does not need to be escaped in XML. Two symbols that **may** be escaped are **single-quote** and **double-quote**. The set of Built-in Entity References are:

Character	Reference	Comment
<	<	Must
>	>	Compatibility with SGML
&	&	Must
'	'	May (IE does not)
"	"	May

Others may be declared in a DTD.

Character References

XML defines the characters available for text as a subset of ISO/IEC 10646 (Unicode) supporting UTF-8 and UTF-16 encoding. Character references allow you to specify an ISO/IEC 10646 character code. A character reference is preceded by:

- **&#** if specified in decimal
- **&#x** if specified in hexadecimal

It is terminated by ; as for entity references. Here are some example character references:

- **£** is the same as £
- **£** is the same as £
- **<** is the same as <
- **<** is the same as <
- **&** is the same as &
- **&** is the same as &

This does give an alternative way of writing the reserved characters.

Signalling the Character set in Use

An XML Processor is required to read UTF-8 and UTF-16 encodings. **UTF** stands for **UCS Transformation Format**. **UCS** stands for**Universal Character set**. UTF-8 is a variable length encoding of Unicode using 1 to 4 bytes per character.The ACSII set of characters are each a single byte. Most non-ideographics are defined in 2 bytes. UTF-16 uses 2 bytes minimum per character.

Other encodings may be used by XML in which case the XML declaration at the head of the file is required. For example:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

ISO-8859 is the old ECMA standard and is a subset of Unicode. ISO-8859-1 (Latin 1) covers all the characters used in the Western European Languages apart from the euro sign! Microsoft Windows supports Latin-1 plus some extensions including the euro.

Discovering the Character Set in Use

The XML encoding declaration may be:

```
<?xml version="1.0" ?>
```

The first four characters can give a good clue as to the encoding being used. In consequence, XML Processors are likely to **sniff** the declaration to try and deduce the character codes in use if not stated explicitly.

2.6 Valid XML

All XML processors must check for well-formed XML. Validating XML requires a Document Type Definition and/or an XML Schema Declaration. A valdating XML parser will use this to detect errors in usage. Validated XML will be discussed as part of Document type Defnitions.

3. XML Examples

3.1 Book Example

A good way to get started with XML is to consider a few examples. Let us suppose that we wish to catalogue our book collection. Let us consider a set of books on XML that might be marked up as follows:

```
<books>
  <book>
    <title>XML IE5</title>
    <author>Alex Homer</author>
    <price>27.49</price>
  </book>
  <book>
    <title>XML Design and Implementation</title>
    <author>Paul Spencer</author>
    <price>36.99</price>
  </book>
  <book>
    <title>XML in Action</title>
    <author>William J. Pardi</author>
    <price>37.49</price>
  </book>
  <book>
      <title>XML: A Primer</title>
    <author>Simon St. Laurent</author>
    <price>23.99</price>
  </book>
  <book>
    <title>XSLT &amp;amp; XPath</title>
    <author>John Robert Gardner and Zarella L. Rendon</author>
    <price>45.99</price>
  </book>
  <book>
    <title>The XML Companion</title>
    <author>Neil Bradley</author>
    <price>24.95</price>
  </book>
</books>
```

Parsing the XML would generate a tree similar to the one in Figure 3.1.

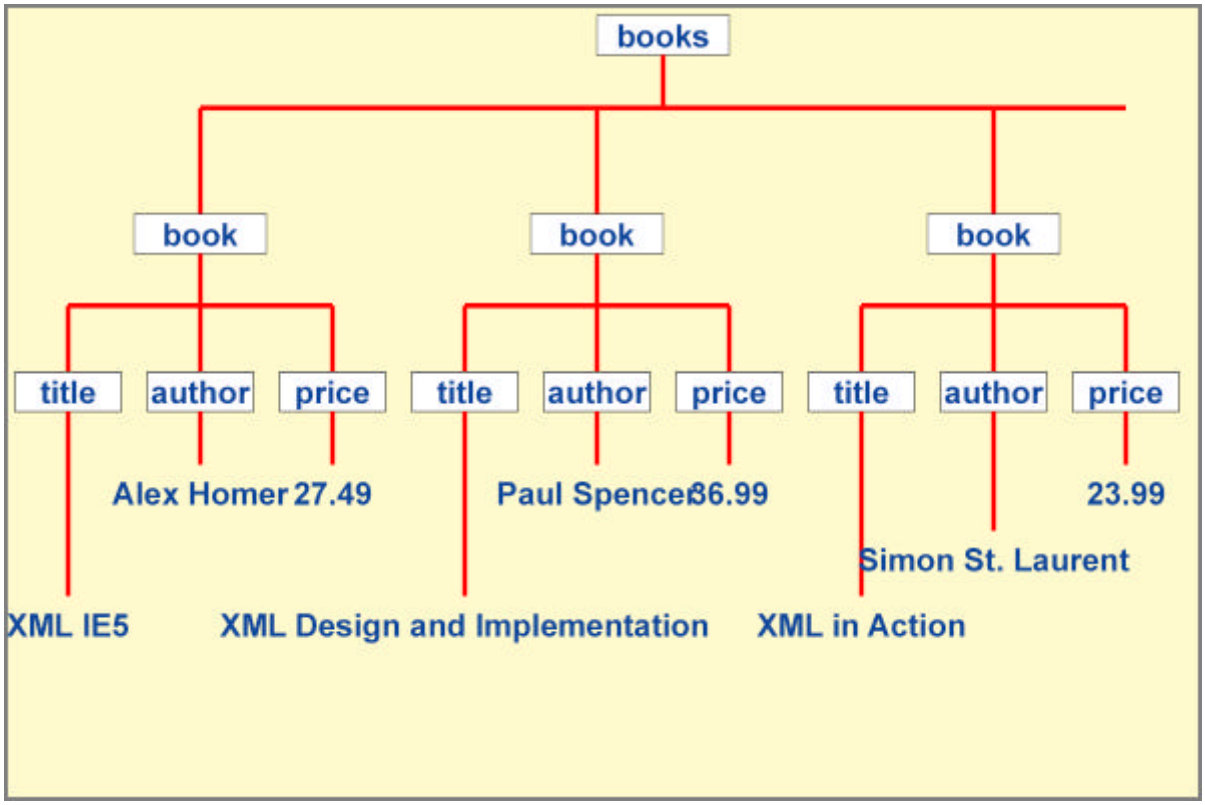

Figure 3.1: Block level and Inline Elements

In the example, the choice has been made to define each piece of information as a separate element. An alternative approach would have been:

```
<books>
  <book title="XML IE5" author="Alex Homer" price="27.49" />
  <book title="XML Design and Implementation" author="Paul Spencer"
price="36.99" />
  <book title="XML in Action" author="William J. Pardi" price="37.49" />
  <book title="XML: A Primer" author="Simon St. Laurent" price="23.99" />
  <book title="XSLT &amp;amp; XPath" author="John Robert Gardner and Zarella L.
Rendon" price="45.99" />
  <book title="The XML Companion" author="Neil Bradley" price="24.95" />
</books>
```

The question is which is the best approach or should it be a mixture of the two?

3.2 Design Issues

Using elements raises the following questions:

- Multiple authors are not handled too well. There is just a long string and some parsing would need to be done to get hold of a single author
- There is the question of whether the elements title, author and price can be in any order. Both are well formed but one might be more desirable.
- What is the unit of currency. It is feasible that the books were purchased using different currencies.
- Given the way the authors have been typed with first name first, it would be difficult to get books in alphabetical order by author surname without some post processing.
- Why elements and not attributes?

One argument against the use of attributes arises in the case of the authors. There probably needs to be some substructure. For example, an element **authors** might be defined enclosing a set of **author** elements. While this would be a simple extension to the element representation, it would be difficult to produce that substructure as an attribute value without making the string cumbersome:

```
<book title="XSLT and XPath"
authors="(John Robert Gardner) , (Zarella L. Rendon)" price="£45.99" />
```

There is also a problem with the book whose title contains &. You cannot write & or &. A good design rule is not to use attributes if that leads to having to construct a parser for the attribute value. The same problem arises if a £ sign is added to the price. so some guidance might be:

- If in doubt, use elements not attributes
- Don't use attributes that need their values parsing
- XML is a lightweight parser, use it
- Restrict attributes to information that is unique and tightly bound to the element and its value
- Difficult to style attributes using CSS
- Try and think what questions may be asked
- Wil it be a human or a machine?
- Don't construct **badly** designed XML

An example of badly designed XML is the one below where there was a need to introduce some markup to indicate what has changed in a document. This could be done by pairs of empty elements as follows:

```
<para>This is a paragraph <startrev />that I have added a phrase too.</para>
<para>This is a new paragraph<endrev /></para>
<para>My original second paragraph</para>
```

This is badly designed. The parsing of the XML would not yield a structure that kept the changed part as a single semantic unit. Rather more verbose but better structured would be:

```
<para>This is a paragraph <rev >that I have added a phrase too.</rev></para>
<para><rev >This is a new paragraph</rev></para>
<para>My original second paragraph</para>
```

3.3 Extending the Book Example

Below is an extension to the book example. Each book has been given its unique ISBN number and this is appropriate as an attribute. The author names have been made into two elements so that getting at the surnames will be easier. A subtitle option has been provided and the publisher has been included. The price now gives the currency value as an attribute of price. This is unique so again might be a sensible attribute:

```
<book isbn="0-13-040446-2">
  <title>XSLT &amp; XPath</title>
  <subtitle>A Guide to XML Transformations</subtitle>
  <authors>
    <author>
      <surname>Gardner</surname>
      <forenames>John Robert</forenames>
    </author>
    <author>
      <surname>Rendon</surname>
      <forenames>Zarella L.</forenames>
    </author>
  </authors>
  <price currency="GBP">45.99</price>
  <publisher>Prentice-Hall</publisher>
  <date>2002</date>
</book>
```

3.4 A Staff Directory

Below is a typical entry in the Rutherford Appleton Laboratory Staff Directory:

```
Matthews, BM (Brian) Dr
ITD
Email: B.M.Matthews@rl.ac.uk
Fax: RAL+5831
R1 1.72 RAL+6648
```

Marking this up in XML might be done as follows:

```
<Employee>
<Name>
  <Surname>Matthews</Surname>
  <Firstname>Brian</Firstname>
  <Firstname>Martin</Firstname>
</Name>
<Title>Dr</Title>
<Dept>ITD</Dept>
<Email>B.M.Matthews@rl.ac.uk</Email>
<Tel code="RAL">6648</Tel>
<Fax code="RAL">5831</Fax>
<Building site="RAL">1</Building>
<Room>1.72</Room>
</Employee>
```

In this case the decision has been made to separate out all the forenames into separate elements. This would be useful if, for example, there were staff members called John Albert Smith, John Maynard Smith, John Robert Smith, etc. The Laboratory has more than one site so the attributes site and code have been used to define the location and which telephone exchange the number referes to.

3.5 CD Collection

Here is a possible way of marking up a CD Collection. The label for this particular CD is unknown and so has been left blank which is perfectly acceptable. Note the work element is the only one where position is important as the tracks have been listed in the order they appear:

```
<cd>
<artist>Aatabou, Najat</artist>
<title>The Voice of the Atlas</title>
<label></label>
<catalog>CDORBD 069</catalog>
<time>61.15</time>
<filed>C05 World</filed>
<playlist>
<work>Baghi narajah</work>
<work>Finetriki</work>
<work>Shouffi rhirou</work>
<work>Lila ya s'haba</work>
<work>Ouardatte lajnane</work>
<work>Ditih</work>
</playlist>
</cd>
```

3.6 SVG

A real world example of an XML Application is Scalable Vector Graphics (SVG). Here considerable used has been made of attributes. The only content that is not defined by attributes is the inner text associated to an element.

```
<svg viewBox="0 0 1023 640" xmlns:xlink="http://www.w3.org/1999/xlink" >

<rect class="background" width="1023" height="640"/>

<g style="stroke:none; fill:lime;"
transform="translate(145, 112) rotate(20) translate(-145, -112)">

<path d="M 0 112c40 48 120-32 160-6c0 0 5 4 10-3c10-103 50-83 90-42c0 0 20
12 30 7c-2 12-18 17-40 17c-55-2-40 25-20 35c30 20 35 65-30 71c-50 4-170 4-
200-79 z"/>

</g>

</svg>
```

4. Styling

4.1 Styling XML

To **present** XML to the user, you need to **style** it. One possibility is Cascading Style Sheets (CSS). CSS provides **linear styling** in the same way as you provide styling to HTML. Styling attributes is not easy until you get to CSS3 so styling is more appropriate when you have defined a set of elements as the markup. Linking a stylesheet to an XML file requires a Processing Instruction which indicates the agent that will do the styling. The simplest example is CSS.

4.2 CSS

The Processing Instruction that links a CSS style sheet to a document has the form:

```
<?xml-stylesheet type="text/css" href="URL" ?>
```

You can add several style sheets and they concatenate just like **@import** in CSS. It is also possible for the Processing Instruction to point to an embedded style sheet:

```
<?xml-stylesheet type="text/css" href="#stylesheet" ?>
- - -
<style id="stylesheet">
- - -
</style>
```

A possible style sheet for our book example might be the file ie.css which has the following contents:

```
books {background-color:navajowhite;display:block}
book {margin:5pt;display:block}
title {font-size:20pt;color:red;font-weight:bold;display:inline}
author{font-size:18pt;color:blue;display:inline}
price {display:none}
```

The Book File itself would contain a reference to the style sheet:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="ie.css"?>
<books>
<book>
<title>XML IE5</title>
<author>Alex Homer</author>
<price>27.49</price>
</book>
. . .
</books>
```

Note that there is a need to indicate what type of element each XML element is. The styling display:block indicates it is a block-level element while display:inline indicates it is an inline element. A value of display:none indicates it is not to be displayed at all. The styling would look something like:

XML IE5 Alex Homer

XML Design and Implementation Paul Spencer

XML in Action William J. Pardi

XML: A Primer Simon St. Laurent

XSLT & XPath John Robert Gardner and Zarella L. Rendon

The XML Companion Neil Bradley

It would be possible to embed the style sheet in the file as follows:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="#stylesheet"?>
<embedded>
<style id="stylesheet">
<!--
books {background-color:navajowhite;display:block}
book {margin:5pt;display:block}
title {font-size:20pt;color:red;font-weight:bold;display:inline}
author{font-size:18pt;color:blue;display:inline}
price {font-size:16pt;color:green;display:none}
-->
</style>
<books>
<book>
<title>XML IE5</title>
<author>Alex Homer</author>
<price>27.49</price>
</book>
. . .
</books>
</embedded>
```

4.3 Display Property

The possible values are given below. Table handling is supported in Opera but not IE6.

Value	Meaning	Like HTML
inline	inline element	em
block	block level element	p
list-item	list item	li
none	No display	None
run-in	Allows for run-in headers	None
compact	Header in margin	None
marker		
table	Table	table
inline-table	Table does not start on a newline	None
table-row-group	Groups one or more rows	tbody
table-header-group	Groups one or more rows at head	thead
table-footer-group	Groups one or more rows at foot	tfoot
table-row	A row of cells	tr
table-column-group	Groups several columns	colgroup
table-column	A column of cells	col
table-cell	Table cell	th, td
table-caption	Caption	caption

Here is an example where the book list has been defined as an HTML-like list:

```
books {display:block;margin:10pt}
book {display:block;margin:20pt}
book *{display:list-item}
```

This would appear as:

- XML IE5
- Alex Homer
- 27.49

- XML Design and Implementation
- Paul Spencer
- 36.99

- XML in Action
- William J. Pardi
- 37.49

- XML: A Primer
- Simon St. Laurent
- 23.99

- XSLT & XPath
- John Robert Gardner and Zarella L. Rendon
- 45.99

- The XML Companion
- Neil Bradley
- 24.95

CSS, for certain applications, is all that is need to present the XML document and gives all the functionality of HTML and CSS with some extensions.

5. XML Namespaces

5.1 Introduction

The aim of the W3C XML Namespaces standard was to allow elements and attributes within an XML document to come from different markup vocabularies (XML applications) and to unambiguously identify which vocabulary they came from. The momentum behind XML means there are many XML applications and the possibility of using several together arises. Each application must have its own **namespace** and we need to differentiate between different XML namespaces. For each **XML namespace**, there is a collection of names used as element types and attribute names and this needs to be identifed as a unique resource.

Let us look at an example.

5.2 Student Example

|Here is a simple exam paper marked up in XML

```
<exam paper="P08770">
<student>Fred Smith</student>
<qapair>
<question>Who invented the Web?</question>
<answer>Tim Berners-Lee</answer>
</qapair>
<qapair>
<question>What year was the web invented?</question>
<answer>1989</answer>
</qapair>
</exam>
```

The problem is that we need to define an element called **student** for each student and another called **paper** that defines the course it is associated with. But the administration almost certainly has such a database already and let us assume these are also stored in XML.

Student Records

The XML database for student records might be something like:

```
<studentset>
<student_record>
<student>Fred Smith</student>
<age>19</age>
<sex>male</sex>
<nationality>UK</nationality>
</student_record>
- - -
</studentset>
```

Record of Papers

The XML database for Courses and Exam Papers might be:

```
<year_two>
<webtech paper="P08770">
<prereq paper="P08769"/>
<exam_date>15.01.02</exam_date>
</webtech>
- - -
</year_two>
```

Nameset Qualifiers

We have 3 different XML applications and rather than each define the element **student** and the attribute **paper**, it would be useful if there was a single place where they were defined. This is achieved in XML by giving each a separate namespace defined as follows:

- Each namespace has a prefix that is associated with it
- A namespace declaration maps a namespace to a specific prefix
- XML names in the nameset do not have to use the prefix
- Attaching the prefix uniquely defines their nameset
- XML names consist of the prefix followed by colon and the local element tag name

To define the namespace uniquely requires a unique identifier for each namepsace. The easiest way on the Web to do that is by defining it as a URL. These are unique by definition:

```
<tagname xmlns:prefx="http://ral.ac.uk/prefx">
<inner>
- - -
</inner>
</tagname>
```

Here we have defined that the prefix we are going to use at the moment for the namespace specified by the URL http://ral.ac.uk/prefx is going to be prefx. To be very precise as to which namespace tags belong to, the example could be written:

```
<prefx:tagname xmlns:prefx="http://ral.ac.uk/prefx">
<prefx:inner>
- - -
</prefx:inner>
</prefx:tagname>
```

The scope of the namespace element is the body of the element where it is declared.

Exam Paper

our exam example could now be written using the two other namespaces sr and csr defined by:

```
<studentset xmlns:sr="http://brookes.ac.uk/strcs">
<student_record>
<student>Fred Smith</student>
<age>19</age>
<sex>male</sex>
<nationality>UK</nationality>
</student_record>
- - -
</studentset>
```

```
<year_two xmlns:csr="http://brookes.ac.uk/csrcns">
<webtech paper="P08770">
<prereq paper="P08769"/>
<exam_date>15.01.02</exam_date>
</webtech>
- - -
</year_two>
```

The exam paper could then be defined using the **student** element from the **http://brookes.ac.uk/strcs** namespace and the **paper** attribute from the **http://brookes.ac.uk/csrcns** namepsace. the way they are used by the exam paper application is to declare the use locally and define the prefix that will be used locally for each:

```
<pap:exam
xmlns:mysr="http://brookes.ac.uk/strcs"
xmlns:mycsr="http://brookes.ac.uk/csrcns"
xmlns:pap="http://brookes.ac.uk/marking"
mycsr:course="P08770">
<mysr:student>Fred Smith</sr:student>
<pap:qapair>
<pap:question>Who invented the web</pap:question>
<pap:answer>Tim Berners-Lee</pap:answer>
- - -
</pap:qapair>
</pap:exam>>
```

A **default namespace** can be defined by a namespace declaration that does not define a prefix. For example:

```
<books xmlns="http://...">
<book title="fred">
. . .
</book>
<book title="frank">
. . .
</book>
</books>
```

Here, the elements books and book together with any other elements used within the books element belong to the namespace specified if no namespace prefix is given.

However, the default namespace does not apply to attributes. In consequence, the attribute title does not automatically belong to the declared namespace unlike the book element itself.

5.3 Book Example

A second example where namespaces could be useful was our book example where clearly it is sensible to use the currency names defined in an ISO database and the Publishers names declared in a Publishers Database. So we get:

```
<currencysymbols xmlns:currency="http://www.iso.com/4217.dtd">
<currency><symbol>ALL</symbol><country>Albania</country><name>Lek</name></currency>
<currency><symbol>AMD</symbol><country>Armenia</country><name>Dram</name></currency>
<currency><symbol>ATS</symbol><country>Austria</country><name>Schilling</name></curre
<currency><symbol>AUD</symbol><country>Australia</country><name>Dollar</name></curren
...
</currencysymbols>
```

```
<publisherlist xmlns:publisher="http://www.lights.com/publisher.dtd">
<creator>http://www.lights.com/publisher/db/</creator>
<base></base>
<publisher><url>1/781</url><name>010 Publishers</name></publisher>
<publisher><url>3/1443</url><name>101 Publishing</name></publisher>
<publisher><url>6/1436</url><name>1st Affordable
Publishers</name></publisher>
<publisher><url>1/7311</url><name>1st4sport, Coachwise</name></publisher>
...
</publisherlist>
```

The book XML application would then look like:

```
<books
xmlns:currency="http://www.iso.com/4217.dtd"
xmlns:publisher="http://www.lights.com/publisher.dtd">
<book isbn="0-13-040446-2">
  <title>XSLT &amp; XPath</title>
  <subtitle>A Guide to XML Transformations</subtitle>
  <authors>
    <author>
      <surname>Gardner</surname>
      <forenames>John Robert</forenames>
    </author>
    <author>
      <surname>Rendon</surname>
      <forenames>Zarella L.</forenames>
    </author>
  </authors>
  <price>
    <currency:symbol>GBP</currency:symbol>
    <amount>45.99</amount>
  </price>
  <publisher:name>Prentice-Hall</publisher:name>
  <date>2002</date>
</book>
```

Unique Specification of Namespace

The **namespace** URL **may** point to the Document Type Definition for the namespace but it does not have to. All that is required is that the URL uniquely defines the namespace. It could point to an informal description. It could point to nothing as long as the URL is not used for something else

Links

Another example of the use of namespaces is within the definition of XML itself. In HTML, links are an integral part of the language via the **a** element. Every XML application is going to have to define links if it is used on the Web so it would be sensible to have a separate namespace for links and this is what has been done. There is a separate XML application called XLink and the namespace declaration is: **xmlns:xlink="http://www.w3.org/1999/xlink"**. This is used in applications like SVG.

Namespaces will be covered in more detail as part of P08772 in the Spring term.

6. A Bibliography of Different Types of Documents

6.1 Introduction

The examples we have shown so far have had quite a regular structure. In many cases, XML documents are not that simple. Consider the case of a true bibliography containing different types of publications. While it is clear that books have authors, standards documents are usually composed by a set of people and it is not usual to list them as authors. For theses, other information such as the university that the person attended might be of interest. Let us see how this would change the simple structure we introduced for books in Section 3.1.

6.2 Books

There is not a great deal more to say about books so let us have a format as follows.

```
<Book citation="B247">
  <Author>
    <Firstname>Tim</Firstname>
    <Surname>Berners-Lee</Surname>
  </Author>
  <BookTitle>
    <Title>Weaving the Web</Title>
  </BookTitle>
  <Publisher>Orion business</Publisher>
  <Year>1999</Year>
</Book>
```

The only thing new is the citation attribute on the top level element giving it a unique local identifier.

6.3 Standards

Standards originate from several organisations. It may not be necessary to define the name of the author or editor but it is important to give the organisation that produced the standard. in some countries, it is mandatory to apply the standards defined by some international standards bodies but not others.

Most standards have a unique number or URL that specifies them. This is needed as it needs to be clear to which standards apply to a particular object. As standards get updated, the year is also important as it is with books. Finally, standards, particularly web standards, come in a variety of formats depending on whether the standard is to be readon the screen or on paper. It is important to know which copies of a standard are identical. This leads to a format something like:

```
<Standard citation="S832">
  <Title>Extensible Markup Language (XML) 1.0</Title>
  <Organisation>World-Wide Web Consortium</Organisation>
  <Number>REC-xml-19980210</Number>
  <Year>1998</Year>
  <Copy href="http://www.w3.org/TR/1998/REC-xml-19980210" type="text/html"
xml:lang="en"/>
  <Copy href="http://www.w3.org/TR/1998/REC-xml-19980210.pdf"
type="application/pdf" xml:lang="en"/>
  <Copy href="http://www.mintert.com/xml/trans/REC-xml-19980210-de.html"
type="text/html" xml:lang="de"/>
</Standard>
```

6.4 Masters Theses

For Master Theses, the main addition has been the university that awarded the thesis and the date has been more precisely defined by including the month as well as the year. This is less usual for books.

```
<MastersThesis citation="M001">
  <Author>
    <Firstname>Kevin</Firstname>
    <Surname>O'Neill</Surname>
  </Author>
  <Title>From A to Web: moving data on the World-Wide Web</Title>
  <School>University of Sunderland</School>
  <Year>1999</Year>
  <Month>June</Month>
</MastersThesis>
```

6.5 Reports

Technical Reports frequently go through many iterations before the final version is produced. In the case of the W3C, the internal reports go through some precise stages such as Working Draft, Candidate Recommendation, Proposed Recommendation etc. It is important to know which stage such a Report has reached so in this case we add an element that defines the Type of the Report.

```
<TechReport citation="T001">
  <Title>XML Schema Part 1: Structures</Title>
  <Organisation>World-Wide Web Consortium</Organisation>
  <Type>Working Draft</Type>
  <Number>WD-xmlschema-1-20000922</Number>
  <Year>2000</Year>
  <Copy href="http://www.w3.org/TR/2000/WD-xmlschema-1-20000922"
type="text/html" xml:lang="en"/>
</TechReport>
```

Appendix A

References

There are some useful Web sites and books relevant to XML:

1. http://www.w3.org/MarkUp/
   The W3C HTML Web Site which contains up-to-date links to anything relevant to HTML.
2. http://www.w3.org/TR/html4/
   HTML Level 4.01 Recommendation, the latest version of HTML, 24 December 1999.
3. http://www.oasis-open.org/cover/
   Robin Cover's XML Cover Page
   Addison Wesley, 1998.
4. Big Book of World Wide Web Recs, Peter Loshin
   Morgan Kaufman, 2000.
5. Web Protocols and Practice, Balachander Krishnamurthy, Jennifer Rexford
   Addison Wesley, 2001.
6. XML How to Program, Deitel, Deitel, Nieto, Lin and Sadhu
   Prentice Hall, 2000.
7. http://www.w3.org/People/Raggett/tidy/, W3C's Tidy Tool that produces legal HTML and tidies up the document
8. http://validator.w3.org/, W3C's Validation Service for Web Pages
9. http://www.w3.org/TR/xhtml1/
   XHTML Level 1.0 Recommendation, a reformulation of HTML as an XML Application, 26 January 2000

Appendix B

XML Activity Statement (January 1996, updated June 1996 and in 1997)

SGML, XML, and Structured Document Interchange

This is W3C activity statement for SGML, XML, and Structured Document Interchange. It is one of the Architecture Domain activities.

Introduction

Most documents on the Web are stored and transmitted in HTML. HTML is a simple language well-suited for hypertext, multimedia, and the display of small and reasonably simple documents. HTML is based on SGML (Standard Generalized Markup Language), an ISO standard system for defining and using document formats.

SGML makes it possible to define your own formats for your own documents, to handle large and complex documents, and to manage large information repositories. Allowing generic SGML in Web documents would facilitate large-scale commercial Web publishing and make it much easier to apply advanced technologies such as Java to Web documents on the user's desktop. However, the full SGML specification is very expensive to implement and goes beyond the needs of the great majority of Web users.

The XML activity is dedicated to bringing the key benefits of generic SGML to the Web in a manner that is easy to implement and understand while remaining fully compliant with the ISO standard.

The goal of the activity is to enable an ISO-compliant subset of SGML, the Extensible Markup Lanaguage (XML), to be served, received, and processed on the Web. As in the case of HTML, the implementation of XML on the Web will require attention not just to structure and content, but also to the standardization of linking and display functions. Since a key design feature of XML is its clear separation of syntax from other processing behaviors, the explicit standardization of the most important of those behaviors (linking and stylesheets) is a necessary part of the XML activity in order to ensure the vendor- and platform-neutral interoperation of XML documents. As in the case of XML syntax, the standardization of XML linking and stylesheets takes place within the context of existing international text processing standards.

Requirements

Web servers and clients conforming to the relevant standards must be able to exchange generic XML documents in a transparent manner. In particular:

- Web servers with XML content must be able prepare data for transmission. This typically includes the generation of a context wrapper with each XML fragment together with pointers to an associated Document Type Definition (DTD) and one or more stylesheets.
- Clients that process XML must be able to unpackage the fragment, parse it in context according to a DTD if required to do so by the document, render it (if a rendering application) in accordance with a specified stylesheet, and correctly interpret hypertext semantics (links, etc.) associated with various document elements.

Products

The specific deliverables of the SGML WG are being developed in three phases, as follows:

- **Phase I: A specification for the syntax of XML (Extensible Markup Language), a simplified version of SGML (ISO 8879) suitable for Internet applications.** The latest working draft of WD-xml-lang was released in March 1997; this is also available in a compressed PostScript version. A Japanese translation of WD-xml-lang has been made by Fuji Xerox. The next revision of WD-xml-lang is scheduled to be released June 30, 1997.
- **Phase II: A specification of standard hypertext mechanisms for XML applications based on HyTime (ISO/IEC 10744) and the Guidelines of the Text Encoding Initiative.** An initial draft of the xml-link spec was presented at the WWW6 Conference (Santa Clara, April 1997). This is also available in a compressed PostScript version. A Japanese translation of WD-xml-link has been made by Fuji Xerox. The next draft revision of WD-xml-link is scheduled to be released June 30, 1997.
- **Phase III: The specification of a standard stylesheet language for XML publishing applications based on DSSSL (ISO/IEC 10179)** together with public text and extensions needed to apply the DSSSL stylesheet language to Web browsers. Target delivery: a draft (WD-xml-style) to be delivered at the SGML/XML 97 Conference in Washington, D. C., December 1997.

Current situation

A paper titled "XML, Java, and the Future of the Web" gives one view of the kind of advanced Web applications made possible by XML; this paper is also available in a compressed PostScript version that demonstrates the application of DSSSL.

Accomplishments of this activity so far include:

- Formation of an SGML Working Group that coordinates with existing related standards efforts and provides specifications where needed to form a complete SGML Internet solution.
- Delivery of a public report on the generic SGML activity within the W3C at the Seybold Conference in San Francisco, September 1996.
- Completion of the first working draft of the syntactic component of XML, a subset of SGML designed for Internet applications.
- Presentation of the XML draft at the SGML 96 Conference in Boston, November 1996.
- Formation of the xml-dev mailing list for XML developers hosted by Imperial College, London.
- Creation of an XML FAQ maintained by Peter Flynn.
- Presentation by the Graphic Communications Association of the first XML Conference in San Diego, March 1997.
- Publication of the revised version of Part 1: Syntax in March 1997.
- Publication of the first draft of Part 2: Linking in April 1997 at the WWW6 Conference in Santa Clara.
- Approval of a Technical Corrigendum to ISO 8879:1986 to align features of XML with the SGML standard at the May 1997 meeting of ISO/IEC JTC1/SC18/WG8 in Barcelona.

Next step

Activities within the W3C

The SGML ERB and WG are currently completing Phases I and II of the XML activity described above.

Requirements for certain enhancements to XML syntax (structured attributes, alternative schemas, multiple name spaces) requested by other W3C activities are under currently under review, and these features may become part of the XML 1.0 working draft before it is submitted for approval as a W3C recommendation.

On July 1, 1997, the group currently known as the W3C SGML Editorial Review Board will become known as the W3C XML Working Group and will begin working under the process rules currently governing the activities of W3C working groups. On the same date, the group currently known as the W3C SGML Working Group will become known as the W3C XML Interest Group and will begin working under the process rules currently governing the activities of W3C interest groups.

Activities outside the W3C

Related efforts are currently taking place outside the W3C:

- Work on transmission of SGML fragments is well advanced within a technical committee of SGML Open, the consortium for SGML tools vendors.
- SGML Open has also specified a catalog mechanism for managing SGML document entities.
- ISO/IEC JTC1/SC18/WG8 has approved a Technical Corrigendum to ISO 8879:1986 in order to reconcile certain user requirements of XML with the SGML standard. That corrigendum has been forwarded to ISO for formal balloting.

Appendix C

XML, Java, and the future of the Web

Jon Bosak, Sun Microsystems

Introduction

The extraordinary growth of the World Wide Web has been fueled by the ability it gives authors to easily and cheaply distribute electronic documents to an international audience. As Web documents have become larger and more complex, however, Web content providers have begun to experience the limitations of a medium that does not provide the extensibility, structure, and data checking needed for large-scale commercial publishing. The ability of Java applets to embed powerful data manipulation capabilities in Web clients makes even clearer the limitations of current methods for the transmittal of document data.

To address the requirements of commercial Web publishing and enable the further expansion of Web technology into new domains of distributed document processing, the World Wide Web Consortium has developed an Extensible Markup Language (XML) for applications that require functionality beyond the current Hypertext Markup Language (HTML). This paper [0] describes the XML effort and discusses new kinds of Java-based Web applications made possible by XML.

Background: HTML and SGML

Most documents on the Web are stored and transmitted in HTML. HTML is a simple language well suited for hypertext, multimedia, and the display of small and reasonably simple documents. HTML is based on SGML (Standard Generalized Markup Language, ISO 8879), a standard system for defining and using document formats.

SGML allows documents to describe their own grammar -- that is, to specify the tag set used in the document and the structural relationships that those tags represent. HTML applications are applications that hardwire a small set of tags in conformance with a single SGML specification. Freezing a small set of tags allows users to leave the language specification out of the document and makes it much easier to build applications, but this ease comes at the cost of severely limiting HTML in several important respects, chief among which are extensibility, structure, and validation.

- *Extensibility.* HTML does not allow users to specify their own tags or attributes in order to parameterize or otherwise semantically qualify their data.
- *Structure.* HTML does not support the specification of deep structures needed to represent database schemas or object-oriented hierarchies.
- *Validation.* HTML does not support the kind of language specification that allows consuming applications to check data for structural validity on importation.

In contrast to HTML stands generic SGML. A generic SGML application is one that supports SGML language specifications of arbitrary complexity and makes possible the qualities of extensibility, structure, and validation missing from HTML. SGML makes it possible to define your own formats for your own documents, to handle large and complex documents, and to manage large information repositories. However, full SGML contains many optional features that are not needed for Web applications and has proven to have a cost/benefit ratio unattractive to current vendors of Web browsers.

The XML effort

The World Wide Web Consortium (W3C) has created an SGML Working Group to build a set of specifications to make it easy and straightforward to use the beneficial features of SGML on the Web. See the W3C SGML Activity page [1] for the current status of this effort. The goal of the W3C SGML activity is to enable the delivery of self-describing data structures of arbitrary depth and complexity to applications that require such structures.

The first phase of this effort is the specification of a simplified subset of SGML specially designed for Web applications. This subset, called XML (Extensible Markup Language), retains the key SGML advantages of extensibility, structure, and validation in a language that is designed to be vastly easier to learn, use, and implement than full SGML.

XML differs from HTML in three major respects:

1. Information providers can define new tag and attribute names at will.
2. Document structures can be nested to any level of complexity.
3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation.

XML has been designed for maximum expressive power, maximum teachability, and maximum ease of implementation. The language is not backward-compatible with existing HTML documents, but documents conforming to the W3C HTML 3.2 specification can easily be converted to XML, as can generic SGML documents and documents generated from databases.

An initial working draft for XML 1.0 [2] has been released for public discussion. A complete specification that includes methods for associating hypertext linking and stylesheet mechanisms with XML documents is scheduled for release at the Sixth World Wide Web Conference in April, 1997.

Web applications of XML

The applications that will drive the acceptance of XML are those that cannot be accomplished within the limitations of HTML. These applications can be divided into four broad categories:

1. Applications that require the Web client to mediate between two or more heterogeneous databases.
2. Applications that attempt to distribute a significant proportion of the processing load from the Web server to the Web client.
3. Applications that require the Web client to present different views of the same data to different users.
4. Applications in which intelligent Web agents attempt to tailor information discovery to the needs of individual users.

The alternative to XML for these applications is proprietary code embedded as "script elements" in HTML documents and delivered in conjunction with proprietary browser plug-ins or Java applets. XML derives from a philosophy that data belongs to its creators and that content providers are best served by a data format that does not bind them to particular script languages, authoring tools, and delivery engines but provides a standardized, vendor-independent, level playing field upon which different authoring and delivery tools may freely compete.

Database interchange: the universal hub

A paradigmatic example of this first category of XML applications is the information tracking system for a home health care agency.

Home health care is a major component of America's multibillion-dollar medical industry that continues to increase in importance as the health care burden is shifted from hospitals to home care settings. Information management is critical to this industry in order to meet the record-keeping requirements of the federal agencies and health maintenance organizations that pay for patient care.

The typical patient entering a home health care agency is represented to the information system by a large collection of paper-based historical materials in the form of patient medical histories and billing data from a variety of doctors, hospitals, pharmacies, and insurance companies. The biggest task in getting the patient into the system is the manual entry of this material into the agency's database.

The coming of the Web has given the medical informatics community the hope that an electronic means can be found to alleviate this burden. Unfortunately, existing Web applications represent fundamentally insufficient models for an adequate solution. Hospitals have begun to offer the agencies a solution that goes something like this:

1. Log into the hospital's Web site.
2. Become an authorized user.
3. Access the patient's medical records using a Web browser.
4. Print out the records from the browser.
5. Manually key in the data from the printouts.

The knowledgeable reader may smile at this "solution," but in fact this is not a joke; this is an actual proposal from a large American hospital known for its early adoption of advanced medical information systems.

A slightly more sophisticated version of this "solution" envisions the operator reading the patient data from the Web browser and keying it directly into the agency's online forms-based interface in a separate window instead of making a printout first. The only difference between this version and the previous one is that it saves the paper that would have been needed for the printout. It does nothing to address the root of the problem. A real solution would look more like this:

1. Log into the hospital's Web site.
2. Become an authorized user.
3. Access the patient's medical records in a Web-based interface that represents the records for that patient with a folder icon.
4. Drag the folder from the Web application over to the internal database application.
5. Drop it into the database.

However, this solution is not possible within the limitations of HTML, for three reasons.

- The HTML tag set is too limited to represent or differentiate between the multitude of database fields in the mixture of documents making up the patient's medical history.
- HTML is incapable of representing the variety of structures in those documents.
- HTML lacks any mechanism for checking the data for structural validity before the receiving application attempts to import it into the target database.

One technically feasible way to implement seamless interchange of patient care records is simply to require all hospitals and health care agencies to use a single standard system dictated by the government (such an approach has actually been suggested). In an environment where hospitals are going out of business on a daily basis and many health care agencies are in deep financial difficulty, however, a scheme that would require them to replace their existing heterogeneous systems with a single new system *en masse* is hardly practical.

The other way to enable interchange between heterogeneous systems is to adopt a single industry-wide interchange format that serves as the single output format for all exporting systems and the single input format for all importing systems. This is, in fact, the purpose for which SGML was initially designed, and XML simply carries on this tradition.

A number of industries, including the aerospace, automotive, telecommunications, and computer software industries, have been using hub languages to perform data interchange for years, and by this time the process is well understood. Typically, the major players in an industry form a standards consortium tasked with defining a Document Type Definition, which is the way in which the tag set and grammar of a markup language are defined. This DTD can then be sent with documents that have been marked up in the industry standard language using off-the-shelf editing tools, and any standard application on the receiving end can validate and process them.

The XML solution is system-independent, vendor-independent, and proven by over a decade of SGML implementation experience. XML merely extends this proven approach to document interchange over the Web. Interestingly, the same day on which the first XML 1.0 draft was released also saw the formal announcement of an SGML initiative within HL7, the standards organization for health care IS vendors, to develop a Health Care Markup Language designed to solve exactly the kind of problem described in this example.

Previous vertical-industry efforts have shown that capturing data in a rich markup often has benefits beyond the immediate requirements of data exchange. In a well-designed standardized patient data system, for example, specific information originally gathered in the course of a routine physical exam and tagged <allergies>, <drug-reactions>, and so on would instantly be available to alert the staff of an emergency room that an unconscious patient from a distant city was allergic to penicillin. The ability of XML to define tags specific to an area of application is critical to this scenario, because the otherwise unqualified word "penicillin" in the thousands of pages of a patient's entire medical history could not trigger the recognition that the same word inside an <allergies> element could trigger.

The health care example is relevant not only because of the scope of the problem and the enormous sums of money involved but also because it is paradigmatic of a very wide range of future Web applications -- any in which Web clients (or Java applications running on those clients) are expected to mediate the lossless exchange of complex data between systems that use different forms of data representation in a way that can be standardized across an industry or other interest group. Some random examples of such applications are:

- Legal publishing
- The government drug approval process
- Collaborative CAD/CAM efforts
- Collaborative calendar management across different systems
- Any corporate network application that works across databases, especially where policies must be enforced: purchase orders, expense requests, etc.
- Exchange of information between players in any broker-organized business: insurance, securities, banking, etc.

Distributed processing: giving Java something to do

A paradigmatic example of this second category of XML applications is the data delivery system designed by the semiconductor industry.

Each major semiconductor manufacturer maintains several terabytes of technical data on all of the ICs that it produces. To enable interchange of this data, an industry consortium (the Pinnacles Group) was formed several years ago by Intel, National Semiconductor, Philips, Texas Instruments, and Hitachi to design an industry-specific SGML markup language. The consortium finished that specification in 1995, and its member companies are now well into the implementation phase of the process.

One might think that the rise in popularity of HTML would cause the Pinnacles members to reconsider their decision, but in fact the limitations of HTML have convinced them that their original strategy was the correct one. Their initial idea was that the richly parameterized data stream made possible by the industry-specific SGML markup would enable intelligent applications not merely to display semiconductor data sheets as readable documents but actually to drive design processes. It is now recognized that this approach is a perfect fit with the concept of distributed Java applets, and the vision of the near future is one in which engineers can access a manufacturer's Web site and download not only viewable data on particular integrated circuits but also a Java applet that allows them to model those circuits in various combinations.

The semiconductor application is a good demonstration of the advantages of XML because:

1. It requires industry-specific markup that cannot be implemented within the confines of the fixed HTML tag set.
2. It requires that the data representation be platform- and vendor-independent so that data from a variety of sources can be used to drive a variety of distributed applications (some of which may be provided by third parties, generating a subindustry of providers of tools that can work with the standardized data stream).
3. Its utility rests ultimately in the fact that a computation-intensive process (modeling circuits for hours at a time) that would otherwise entail an enormous, extended resource hit on the server has been changed into a brief interaction with the server followed by an extended interaction with the user's own Web client. This aspect has been summed up in the slogan "XML gives Java something to do."

Note that validation, while sometimes important, does not always play the crucial role in this category of applications that it does in applications where data must be checked for structural integrity before entering a database. To make processing as efficient as possible, XML has been designed so that validation is optional in applications where it is not needed.

As with the health-care example, the semiconductor application is notable not merely for the sheer size of the market it represents but also because it is paradigmatic of an enormous range of future Java-based Web applications -- virtually any application in which standardized data is expected to be manipulated in interesting ways on the client. Perhaps the most obvious examples of such applications are the following:

- Design applications where the designer would otherwise use server cycles to consider various alternatives: electronics, engineering, architecture, menu planning, etc.
- Scheduling applications where a customer would otherwise use server cycles to entertain various possibilities: airlines, trains, buses, and subways; restaurants, movies, plays, and concerts. This is what Easy Saabre and Ticketron will look like a few years from now as the economies of distributed Java-based processing become evident.
- Commercial applications that allow consumers to explore alternatives by supplying different shopping criteria: real estate, automobiles, appliances, etc.
- The entire spectrum of educational applications, a small subset of which are the ones we call "online help".
- The entire spectrum of customer-support applications, ranging from lawn-mower maintenance through technical support for computers.

A harbinger of applications to come in the last category is the Solution Exchange Standard, an SGML markup language announced last June by a consortium of over 60 hardware, software, and communications companies to facilitate the exchange of technical support information among vendors, system integrators, and corporate help desks. In the words of the announcement:

The standard has been designed to be flexible. It is independent of any platform, vendor or application, so it can be used to exchange solution information without regard to the system it is coming from or going to. [...] Additionally, the standard has been designed to have a long lifetime. SGML offers room for growth and extensibility, so the standard can easily accommodate rapidly changing support environments.

Such applications, which the XML subset is specifically designed to address, will grow in importance as consumers come to expect interoperability among their data-manipulating applets and information providers confront the realities of trying to support computation-intensive tasks directly on their Web servers.

View selection: letting the user decide

A third variety of XML applications are those in which users may wish to switch between different views of the data without requiring that the data be downloaded again in a different form from the Web server.

One early application in this category will be dynamic tables of contents. It is possible now, using Web servers built on object-oriented databases, to present the user with a table of contents into a large collection of data that can be expanded with a mouse click to "open up" a portion of the TOC and reveal more detailed levels of the document structure. Dynamic TOCs of this kind can be generated at run time directly from the hierarchical structure of the document. Unfortunately, the Web latency built into every expansion or contraction of the TOC makes this process sluggish in many user environments. A much better solution is to download the entire structured TOC to the client rather than just individual server-generated views of the TOC. Then the user can expand, contract, and move about in the TOC supported by a much faster process running directly on the client.

A group at Sun actually implemented a form of this solution as part of a Java-based HTML help browser, but the limitations of HTML required the team to come up with a couple of clever workarounds. In this application, a TOC was constructed by hand (the lack of structure in ordinary HTML makes it impossible to reliably generate a TOC directly from the document) using nonstandard tags invented for the purpose, and then the TOC piece was wrapped in a comment within an HTML page to hide the nonstandard markup from Web browsers. A Java applet downloaded with the HTML document interpreted the hidden markup and provided the client-based TOC behavior.

In practice, this application worked very well and testified both to the ingenuity of its designers and to the validity of the basic concept. But in an XML environment, neither the manual creation of the TOC nor its concealment would have been necessary. Instead, standard XML editors would have been used to create structured content from which a structured TOC could be generated at run time and downloaded to browsers that would automatically create and display the TOC using either a downloaded Java applet or a standard set of JavaHelp class libraries.

The ability to capture and transmit semantic and structural data made possible by XML greatly expands the range of possibilities for client-side manipulation of the way data appears to the user. For example:

- A technical manual that covers both the Sparc and x86 versions of the Solaris operating system can be made to appear like a manual for Sparc only, or a manual for x86 only, just by clicking a preferences switch.
- An installation sheet that carries warnings in multiple languages can be made to show just the ones in the language selected by the user.
- A document containing many annotations can be switched from a mode that shows only the text, to a mode that shows only the annotations, to a mode that shows both, just by making a menu selection.
- A phone book sorted by last name can instantly be changed into a phone book sorted by first name.

This list only hints at the possible uses that creative Web designers will find for richly structured data delivered in a standardized way to Web clients.

Web agents: data that knows about me

A future domain for XML applications will arise when intelligent Web agents begin to make larger demands for structured data than can easily be conveyed by HTML. Perhaps the earliest applications in this category will be those in which user preferences must be represented in a standard way to mass media providers. The key requirements for such applications have been summed up by Matthew Fuchs of Disney Imagineering: "Information needs to know about itself, and information needs to know about me."

Consider a personalized TV guide for the fabled 500-channel cable TV system. A personalized TV guide that works across the entire spectrum of possible providers requires not only that the user's preferences and other characteristics (educational level, interest, profession, age, visual acuity) be specified in a standard, vendor-independent manner -- obviously a job for an industry-standard markup system -- but also that the programs themselves be described in a way that allows agents to intelligently select the ones most likely to be of interest to the user. This second requirement can be met only by a standardized system that uses many specialized tags to convey specific attributes of a particular program offering (subject category, audience category, leading actors, length, date made, critical rating, specialized content, language, etc.). Exactly the same requirements would apply to customized newspapers and many other applications in which information selection is tailored to the indvidual user.

While such applications still lie over the horizon, it is obvious that they will play an increasingly important role in our lives and that their implementation will require XML-like data in order to function interoperably and thereby allow intelligent Web agents to compete effectively in an open market.

Advanced linking and stylesheet mechanisms

Outside XML as such, but an integral part of the W3C SGML effort, are powerful linking and stylesheet mechanisms that go beyond current HTML-based methods just as XML goes beyond HTML.

Linking

Despite its name and all of the publicity that has surrounded HTML, this so-called "hypertext markup language" actually implements just a tiny amount of the functionality that has historically been associated with the concept of hypertext systems. Only the simplest form of linking is supported -- unidirectional links to hardcoded locations. This is a far cry from the systems that were built and proven during the 1970s and 1980s.

In a true hypertext system of the kind envisioned for the XML effort, there will be standardized syntax for all of the classic hypertext linking mechanisms:

- Location-independent naming
- Bidirectional links
- Links that can be specified and managed outside of documents to which they apply
- N-ary hyperlinks (e.g., rings, multiple windows)
- Aggregate links (multiple sources)
- Transclusion (the link target document appears to be part of the link source document)
- Attributes on links (link types)

The first draft of a specification for basic standardized hypertext mechanisms to be used in conjunction with XML is scheduled for release at the Sixth World Wide Web Conference in April, 1997.

Stylesheets

The current CSS (cascading style sheets) effort provides a style mechanism well suited to the relatively low-level demands of HTML but incapable of supporting the greatly expanded range of rendering techniques made possible by extensible structured markup. The counterpart to XML is a stylesheet programming language that is:

- Freely extensible so that stylesheet designers can define an unlimited number of treatments for an unlimited variety of tags.
- Turing-complete so that stylesheet designers can arbitrarily extend the available procedures.
- Based on a standard syntax to minimize the learning curve.
- Able to address the entire tree structure of an XML document in structural terms, so that context relationships between elements in a document can be expressed to any level of complexity.
- Completely internationalized so that left-to-right, right-to-left, and top-to-bottom scripts can all be dealt with, even if mixed in a single document.
- Provided with a sophisticated rendering model that allows the specification of professional page layout features such as multiple column sets, rotated text areas, and float zones.
- Defined in a way that allows partial rendering in order to enable efficient delivery of documents over the Web.

Such a language already exists in a new international standard called the Document Style Semantics and Specification Language (DSSSL, ISO/IEC 10179). Published in April, 1996, DSSSL is the stylesheet language of the future for XML documents. An initial specification of a DSS SL subset [3] for use with XML applications has already been published. This specification will be further developed as part of the XML activity.

Conclusion

HTML functions well as a markup for the publication of simple documents and as a transportation envelope for downloadable scripts. However, the need to support the much greater information requirements of standardized Java applications will necessitate the development of a standard, extensible, structured language and similarly expanded linking and stylesheet mechanisms. The W3C SGML effort is actively developing a set of specifications that will allow these objectives to be met within an open standards environment.

Acknowledgements

The author would like to thank his colleagues in the Davenport Group for early contributions to the beginnings of this document. The example applications were clarified and expanded with the help of participants in the workshop "Internet Applications of SGML and DSSSL" held at the GCA Information and Technology Week in Seattle on August 23, 1996. Special thanks are due to Tim Bray, Kurt Conrad, Steve DeRose, Matt Fuchs, and Murray Maloney for their outstanding contributions to the workshop.

Production note

This paper was written in HTML 3.2 and formatted by the Jade DSSSL engine [4] for printout. The section numbers, headers, footers, and Table of Contents seen in the printed version are not part of the HTML source [5] but were generated automatically as specified by a DSSSL stylesheet [6].

References

[0] http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.ps.zip
[1] http://www.w3.org/pub/WWW/MarkUp/SGML/activity
[2] http://www.w3.org/pub/WWW/TR/WD-xml-961114.html
[3] http://sunsite.unc.edu/pub/sun-info/standards/dsssl/dssslo/dssslo.htm
[4] http://www.jclark.com/jade/
[5] http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm
[6] http://sunsite.unc.edu/pub/sun-info/standards/dsssl/stylesheets/html32/html32hc. dsl