XML, a new start for the Web

Michel Goossens

Academic Training, May 2000

Contents

PartI
The Extensible Markup Language, an introduction

1 The quest for structured markup

1.1 The 1980s

- Transition from paper (printing) to electronic medium (disk, CDROM);
- drop in cost, and hence ubiquity of the PC;
- enormous increase of data volume (dictionaries, telephone directories, legal texts, image data banks);
- increase in the cost to store and maintain (keep up-to-date) the information;
- old systems were optimised to print or visualise (WYSIWYG);
- there was a clear need to better *structure* the information.

1.2 The birth of SGML

- Documents that are marked up in a structured way can be more easily exchanged and manipulated if they adhere to an *open* standard;
- Gencode (GCA) and GML (IBM) were in the nineteen seventies precursors of a generic markup language;
- based on the experience gained with GML ANSI, and then ISO in 1986 adopted the SGML standard (ISO:8879, 1986);
- SGML is quite a complex standard and needs non-trivial resources to get going (training, expensive software);
- thus very long SGML's deployment was limited to large companies, government agencies and institutions.

1.3 Then HTML took the world by storm

- Since the late 1980s the importance of the Internet in communication has exploded exponentially: TCP/IP was (and still is) its *lingua franca* and *e-mail*, *telnet* and *ftp* the main user-level applications;
- in 1989/90 at CERN there was a real need to give and get access to documents in a transparent and location-independent way;
- thanks to its *Mac/NeXT culture* (simple, rich, and user-friendly interface) Tim Berners-Lee (an Oxford graduate) and collaborators coined the three basic ideas of the Web:
  1. *Hypertext Transport Protocol* (HTTP): communication between WWW servers;
  2. *Uniform Resource Locator* (URL): location-independent addressing scheme for resources on the Web;
  3. *Hypertext Markup Language* (HTML): the markup language.
- This mini Web (in 1991 there are only 10 Web servers in the world!) was a nice toy during the years 1990-1994, especially when you had access to a Mac or Next computer.
- However, a *killer application* was needed to get the Web *out of the lab*. It was Marc Andreessen and Co with their *Mosaic* who provided a nicely integrated graphic user interface to the who got the Web ball rolling;
- In May 1994 at CERN the First WWW Conference was organised (the so called *Woodstock of the Web*) and since then the number of Web users and servers has increased into the millions.
- The W3 Consortium (MIT, Inria, Keio) was set up in 1994-1995.
- Many applications appeared (Netscape, Internet Explorer) and the rest is history.

2 HTML, its problems and the proposed solutions

2.1 What is wrong with HTML anyway?

- *Invalid HTML:* applications produce invalid HTML or introduce vendor-specific extensions; browsers do not reject invalid HTML.
- *Broken links:* problem of deleted or moved Web pages (use URN).
- *Fixed grammar:* HTML is a *fixed* SGML DTD (language syntax); no *standard* way to extend, adapt, renew the language.
- *Limited support for meta-data:* what about keywords, search-engines, re-use of the same source file.
- *Absence of structural tags:* HTML tags control mainly appearance not structure; navigation difficult.
- *Data exchange difficulties:* HTML aimed at presentation (Latin1 codes); difficult for data mining, internationalisation, exchange.
- *Absence of modern features:* Web changes rapidly; HTML must adapt smoothly to new technology (dynamic clients with applets, object data model, etc.).

2.2 What can be done about it?

- *Extend HTML:* HTML 4/4.1 W3C Recommendations.
- *Cascading Style Sheets:* CSS1/CSS2 recommendations to decouple presentation from content.
- *Dynamic HTML (DHTML):* provides functionality to add dynamic representation to Web pages (JavaScript, Visual Basic).
- *Document Object Model (DOM):* allows programs to access HTML (XML) elements as a structured collection of object data, each having a set of properties and methods.
- *Extensible Markup Language (XML):* offers better application-specificity and better data organisation.
- *XHTML:* a redefinition (and modularisation) of HTML 4.x as a series of XML DTDs.

A problem is that at present very few browsers, if any, support these new features

3 XML: A set of Extensible Markup Languages

- Since mid 1996 W3C SGML (now renamed XML) Working Group had been developing a *system for defining, validating, and sharing document formats on the Web.*
- W3C issued XML 1.0 as a Recommendation on the 10th of February 1998.
- It is the first in a series of recommendations.
  - Document Object Model Level 1, 2 (DOM);
  - Mathematical Markup Language (MathML) 1.01;
  - Resource Description Framework (RDF);
  - XPath: addressing parts of an XML document;
  - XSLT: transforming XML documents for use with XSL;
  - Synchronised Multimedia Integration Language (SMIL) 1.0;
  - Namespaces in XML.
- Ongoing work includes:
  - Mathematical Markup Language (MathML) 2.0;
  - XML Schema (description of structures and data types);
  - XHTML: extensible HTML-A Reformulation of HTML 4.0 in XML 1.0;
  - XSL-FO: extensible stylesheet language formatting objects;
  - SVG: scalable vector graphics;
  - XLink: XML Linking Language;
  - XPointer: XML Pointer Language.

Dozens of XML-based languages exist already, and more are created almost dayly.

- WAP: Wireless Application Protocol;
- AML: Astronomical Markup Language;
- BIOML: Biopolymer Markup Language;
- GEML: Gene Expression Markup Language;
- X3D: 3-D graphics, virtual reality (VRML);
- MusiXML: music notation;
- VXML: vocal dialogues;
- CML: Chemical Markup Language;
- FpML: Financial products Markup Language;
- TEI: Text Encoding Initiative;
- DocBook: Computer documentation;
- etc.

4 What is XML?

- light-weight subset of SGML (ISO/IEC 8879:1986);
- meta-language to described logical structure of document;
- well integrated with the Internet;
- internationalisation built in from the start (*document declaration* part permits Unicode);
- easy to learn, modify and implement;
- logical relations between elements described in *Document Type Definition* (DTD);
- documents can be *well-formed* (no DTD needed, nesting correct).
- documents can be *valid* (according to a DTD).

5 XML: A simple example

5.1 Well-formed and valid document

- A document contains one or more elements.
- There is exactly one element, called the root, or document element, no part of which appears in the content of any other element.
- For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

```
1:  <!-- proper nesting -->
2:  <root>This sentence <important>must be read together the
3:  following <super>superword</super></important></root>
4:
5:  <!-- improper nesting -->
6:  <root>This sentence <important>must be read together the
7:  following <super>superword</important></super></root>
```

- a minimal *well-formed* XML document;

```
<welcome>Welcome to CERN in Geneva!</welcome>
```

- this document can be made *valid* by adding a DTD:

```
1:  <?xml version="1.0" standalone="yes"?>
2:  <!DOCTYPE welcome [
3:    <!ELEMENT welcome (#PCDATA)>
4:  ]>
5:  <welcome>Welcome to CERN in Geneva!</welcome>
```

This document has three parts:

1. an XML processing instruction (version, encoding, stand-alone or no);

2. a document type declaration (internal and external subsets);
3. the document instance.

5.2 Characters and encodings

Characters (`Char`) can be part of tags, attribute names, or character data. The Unicode encodings UTF-16 and UTF-8 are recognised implicitly by XML parsers. All other encodings *must* be declared on the XML declaration.

Consider the following little file containing non-ASCII (Latin-1, aka, ISO/IEC 8859-1) characters:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE bonneidée [
  <ELEMENT bonneidée (#PCDATA)>
]>
<bonneidée>XML est une excellente idée !</bonneidée>
```

Note the `encoding="ISO-8859-1"` specification on the first line (the XML declaration). When we feed this file to some of the parsers this is what we get.

```
>xml4j.sh  bonneidee.xml
bonneidee.xml: 345 ms (1 elems, 0 attrs, 0 spaces, 28 chars)

>xp.sh bonneidee.xml
<bonneidée>XML est une excellente idée !</bonneidée>

>nsgmls xml.dcl bonneidee.xml
?xml version="1.0" encoding="ISO-8859-1" standalone="yes"
(bonneidée
-XML est une excellente idée !
)bonneidée
```

5.3 Analysing an XML document by API

5.3.1 SAX interface

SAX 2.0 (*Simple API for XML*) [18] is a standard interface for event-based XML parsing. The event-based API reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree (as apposed to the DOM interface discussed later). The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Consider the following simple XML file `simpledoc.xml`. We shall use it as an example for both the SAX and DOM interfaces.

```
<?xml version="1.0" ?>
<!-- This is a simple XML document -->
<simpledoc>
<title>Using SAX and DOM</title>
<para>This is a some content.</para>
</simpledoc>
```

The SAX standard defines several methods to deal with event. We only show two methods (from a Java implementation) to output some information when the start and end of an element event is signaled is shown below.

```
    public void startElement (String uri, String name,
                              String qName, Attributes atts)
    {
         System.out.println("Start element: {" + uri + "}" + name);
    }

    public void endElement (String uri, String name, String qName)
    {
         System.out.println("End element:   {" + uri + "}" + name);
    }
```

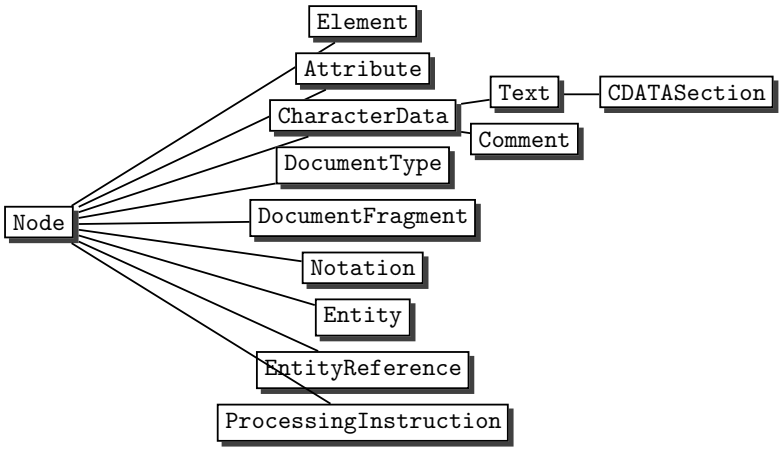

Figure 1: Interface hierarchies for the DOM

We run the example program containing these line with a SAX parser and obtain the following output:

```
> export CLASSPATH=/opt/java/utilities.jar:/opt/java/sax2.jar:.
> java -Dorg.xml.sax.driver=org.brownell.xml.aelfred2.SAXDriver MySAXExa simpledoc.xml
Start document
Start element: {}simpledoc
Characters:    "\n"
Start element: {}title
Characters:    "Using SAX and DOM"
End element:   {}
Characters:    "\n"
Start element: {}para
Characters:    "This is a some content."
End element:   {}
Characters:    "\n"
End element:   {}
End document
```

5.3.2 DOM interface

The DOM (Document Object Model) is a standard internal representation of the document structure.

- easy programming access to components;
- delete, add or edit componenents' content, attributes and style;
- allows writing applications working properly on all browsers and servers, and on all platforms;
- one programming model with multiple computer language bindings (Java, ECMAScript, etc);
- platform- and language-neutral program interface;
- relies on an internal tree-like representation of the document, enabling the document hierachy to be traversed.

At present DOM level 1 [28] (basic core) and level 2 [29] (style sheet object model and features for manipulating the style information attached to a document; traversals on the document; definition of an event model; support for XML namespaces) exist. Work on level 3 (document loading, saving, XML schema, validation) is just starting.

In a system using the DOM the parser analyses (and possibly validates) an XML document and constructs in memory a hierarchical representation of the various document components in the form of objects (elements, attributes, text fragments, etc.). Several methods defined for those objects can then be used by application software (XML editors, Internet browsers) to navigate the document, or manipulate its content.

As for large documents the DOM builds quite large structures in memory it is often preferable for applications where the document tree does not need to be changed to use the simpler event-based SAX interface.

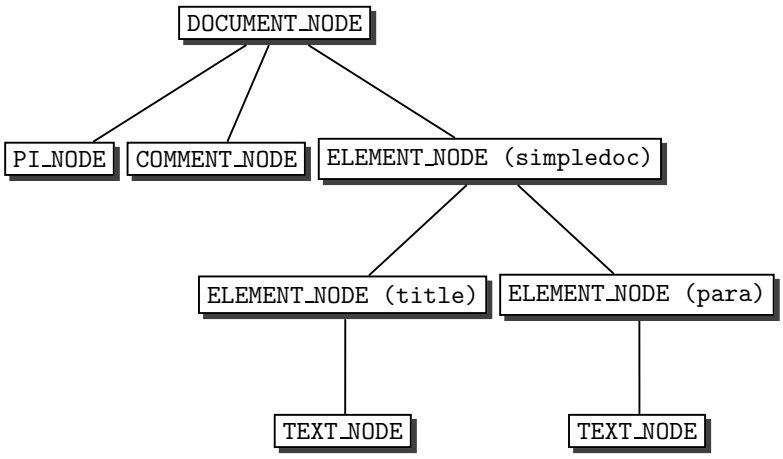

Figure 2: Our simpledoc XML file viewed as a DOM tree

6 Document-level information

6.1 Common Syntactic Constructs

6.1.1 White Space

White space consists of one or more space (`#x20`) characters, carriage returns, line feeds, or tabs.

6.1.2 Names and Tokens

Unicode classifies characters for convenience as *letters*, *digits*, or *other characters*. Letters consist of an alphabetic or syllabic base character possibly followed by one or more combining characters, or of an ideographic character.

A *Name* is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters.

Names beginning with the letters `xml` (in any uppercase/lowercase combinations) are reserved. Names containing the colon character are reserved for specifying the namespace.

6.1.3 Literals

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string.

6.2 Character Data and Markup

Text consists of intermingled character data and markup. Markup takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, `CDATA` section delimiters, document type declarations, and processing instructions.

The ampersand character (&) and the left angle bracket (<) may appear in their literal form only when used as markup delimiters, or within a comment, a processing instruction, or a `CDATA` section. Elsewhere, they must be escaped using either numeric character references or the strings `&` and `<`.

On the other hand, greater-than (>) may be escaped as `>`, the single-quote (') may be written `&apos;`, and the double quote (") may be written `&quote;`.

6.3 Comments

Comments may appear anywhere in a document outside other markup or in the DTD at places allowed by the grammar. They are not part of the document's character data (an XML processor may, but need not,

make it possible for an application to retrieve the text of comments). The string `--` (double-hyphen) must not occur within comments.

An example of a comment is the following:

```
<!-- A <super>super & grandiose</super> comment -->
```

6.4 Processing Instructions

Processing instructions (PIs) allow documents to contain instructions for applications. PIs are not part of the document's character data, but must be passed through to the application. The PI begins with a target (PITarget) used to identify the application to which the instruction is directed.

An example of a processing instruction for `xmltex` would be:

```
<?xmltex \newline ?>
```

6.5 CDATA Sections

`CDATA` sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognised as markup.

An example of a CDATA section is shown below:

```
<![CDATA[Un texte <super>super & grandiose</super>.]]>
```

`CDATA` sections come also in handy when you are writing technical documentation, containing, e.g., C or C++ code fragments:

```
1:  <![CDATA[#include <iostream>
2:  #include <vector>
3:  int array[] = { 1, 42, 3 }; // Regular "C" array.
4:    ...
5:    for ( p1 = array; p1 != array + 3; ++p1 )
6:      cout << "array has " << *p1 << "\n";
7:  ]]>
```

6.6 Prolog and Document Type Declaration

Ideally every XML document should begin with an XML declaration which specifies the version of XML being used.

6.6.1 The prolog

For example, the following is a complete XML document, well-formed but not valid:

```
<?xml version="1.0">
<welcome>Welcome to CERN in Geneva!</welcome>
```

where the number `1.0` indicates the version of XML to which the document conforms.

Note that the following is also well-formed (without XML declaration):

```
<welcome>Welcome to CERN in Geneva!</welcome>
```

The function of the markup in an XML document is to describe its *storage* and *logical structure* and to associate attribute-value pairs with its logical structures. The DTD can define constraints on the logical structure and support the use of predefined storage units. An XML document is *valid* if it has an associated DTD and if the document complies with the constraints expressed in it.

The DTD must appear before the first element in the document. The following turn the previously shown well-formed document into a valid one.

```
1:  <?xml version="1.0" standalone="yes"?>
2:  <!DOCTYPE welcome [
3:    <!ELEMENT welcome (#PCDATA)>
4:  ]>
5:  <welcome>Welcome to CERN in Geneva!</welcome>
```

6.6.2 The document type definition

The XML *document type declaration* contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a *document type definition* (DTD). The DTD can point to an *external subset* (an external entity) containing markup declarations, it can contain markup declarations directly in an *internal subset*, or can do both. The DTD for a document consists of both subsets taken together.

A DTD declares element types declaration, attribute-lists entities, and notations. These declarations may be contained in whole or in part within *parameter entities*.

6.6.3 The external subset

The external subset and external parameter entities differ from the internal subset in that they can contain parameter-entity references in their markup declarations, not only between markup declarations.

For instance we can rewrite the above valid document using an external DTD `welcome.dtd` with the following content.

```
<!ELEMENT welcome (#PCDATA)>
```

The XML file then references this file by specifying its URI [15]:

```
1:  <?xml version="1.0" standalone="yes"?>
2:  <!DOCTYPE welcome SYSTEM "welcome.dtd">
3:  <welcome>Welcome to CERN in Geneva!</welcome>
```

It is important to realize that, if both the external and internal subsets are used, the internal subset is considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset *take precedence* over those in the external subset.

6.7 White space handling

For reasons of readability and maintainability it is often advantageous when editing XML documents to use *white space*, such as spaces, tabs, and blank lines. This white space is typically irrelevant and should not be included in the final version of the document.

There exist occasions (poetry, source code) where white space in the source is *significant* and should be preserved in the final output.

A special attribute `xml:space` exists to indicate that for a given element white space should be preserved by applications.

```
1:  <!ATTLIST poem   xml:space (default|preserve) 'preserve'>
2:  <!ATTLIST code   xml:space (default|preserve) 'preserve'>
```

6.8 End-of-Line handling

XML parsed entities are often stored in computer files which, for editing convenience, are organised into lines that are separated by a combination of carriage-return (`#xD`) and line-feed (`#xA`) characters. To simplify the tasks of applications, XML processors will *normalise* the output by passing to applications a single line-end character `#xA`.

6.9 Language identification

If is often useful to identify the language in which the content is written. A special attribute `xml:lang` is available to specify the language used in the contents and attribute values of any element in an XML document.

```
 1:    <p xml:lang="fr">Cachez ce sein que je ne saurais voir.</p>
 2:    <p xml:lang="fr-FR">Ce programme a une bogue.</p>
 3:    <p xml:lang="fr-CA">Ce programme a un bogue.</p>
 4:    <p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
 5:    <p xml:lang="en-GB">What colour is it?</p>
 6:    <p xml:lang="en-US">What color is it?</p>
 7:      <sp qui="Faust" desc='leise' xml:lang="de">
 8:         <l>Habe nun, ach! Philosophie,</l>
 9:         <l>Juristerei, und Medizin</l>
10:      </sp>
```

The intent declared with `xml:lang` is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of `xml:lang` on another element within that content.

7 Logical structures

Each XML document contains one or more *elements*, the boundaries of which are either delimited by *start-tags* and *end-tags*, or, for *empty elements*, by an empty-element tag. Each element has a *type*, identified by name, (its "generic identifier or GI), and may have a set of *attribute specifications*, which are characterised by a name and a value.

The name in an element's end-tag must match the element type in the start-tag, while for *valid* documents the contents of the element must correspond to its declaration in the DTD.

7.1 Start-tags, end-tags, and empty-element tags

7.1.1 The start-tag

The beginning of every non-empty XML element is marked by a start-tag.

- No attribute name can appear more than once in a start-tag or an empty-element tag.
- For valid documents the attribute must have been declared in the DTD and the value must be of the declared type.
- Attribute values cannot contain direct or indirect entity references to external entities.
- The replacement text of any entity referred to directly or indirectly in an attribute value (other than `<`) must not contain a <.

An example of a start-tag:

```
<line number="1" author='moli&egrave;re' stanza="17">
```

7.1.2 The end-tag

The end of every element that begins with a start-tag must be marked by an end-tag containing a name that echoes the element's type as given in the start-tag:

An example of an end-tag:

```
</poem>
```

7.1.3 Content of Elements

The text between start-tag and end-tag is called the element's content

7.1.4 Empty elements tags

If an element is empty, it must be represented either by a start-tag immediately followed by an end-tag or by an empty-element tag, which takes the following special form.

An example of empty tags are shown below

```
1:  <IMG align="left"
2:       src="http://www.w3.org/Icons/WWW/w3c_home" />
3:       <br></br>
4:       <br/>
```

7.2 Element Type Declarations

For validation purposes one can constrain the element structure of an XML document by using element type and attribute-list declarations. An element type declaration constrains the element's content by specifying which element types can appear as children of the element.

- `Name` specifies the *element type* being declared.
- Element types can only be declared *once*.

A few examples of element declarations follow.

```
1:  <!ELEMENT poème       (préambule, corps)>
2:  <!ELEMENT préambule   (titre, recueil?, auteur, date?)>
3:  <!ELEMENT title       (#PCDATA)>
4:  <!ELEMENT img         EMPTY>
5:  <!ELEMENT %para.name; %para.content; >
6:  <!ELEMENT container   ANY>
```

7.2.1 Element content

An element type has element content when elements of that type must contain only child elements (no character data), optionally separated by white space.

Examples of element-contents models are:

```
1:  <!ELEMENT poème     (préambule, corps, postscript)>
2:  <!ELEMENT préambule (titre, recueil?, auteur, date?)>
3:  <!ELEMENT taxonomy  (category+ | ((bibl | biblFull), category*)) >
4:  <!ELEMENT chap-body (%chap.mix; | %chap.sup | %chap.local)*>
```

7.2.2 Mixed Content

An element type has mixed content when elements of that type may contain character data, optionally interspersed with child elements. In this case, the types of the child elements may be constrained, but not their order or their number of occurrences. The `#PCDATA` declaration is always comes first. A name can only occur once in a single mixed content declaration.

Examples of mixed content declarations:

```
1:  <!ELEMENT para   (#PCDATA|bold|emph)*>
2:  <!ELEMENT figure (#PCDATA | para | includegraphics | caption |
3:                            %inline;|%likepara;)*>
4:  <!ELEMENT code   (#PCDATA)>
```

7.3 Attribute-List Declarations

Attributes, which may appear only within start-tags and empty-element tags, associate name-value pairs with elements. Attributes,

- define the set of attributes pertaining to a given element type;
- establish type constraints for these attributes;
- provide default values for attributes.

7.3.1 Attribute Types

There exist three type of attributes for XML element types:

- a string type;
- a set of tokenized types;
- enumerated types.

7.3.2 Enumerated attribute types

Enumerated attributes can take one of a list of values provided in the declaration. There are two kinds of enumerated types:

7.3.3 Attribute defaults

An attribute declaration provides information on whether the attribute's presence is required, and if not, which action should be taken if the declared attribute is absent in a document.

7.3.4 Examples of attribute declarations

```
<!ENTITY % basic  "id    ID    #IMPLIED
                   class CDATA #IMPLIED
                   style CDATA #IMPLIED">
<!ATTLIST url      name  CDATA #REQUIRED>
<!ATTLIST includegraphics %basic;
                          filename CDATA #REQUIRED
                          width    CDATA #IMPLIED
                          height   CDATA #IMPLIED
                          scale    CDATA ".5"
                          angle    FIXED ".0">
<!ATTLIST list type (bulleted|ordered|gloss) "bulleted">
```

7.3.5 Attribute-Value Normalisation

Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalise it (resolve character references, process the replacement text of entity references, and handle whitespace characters).

7.4 Conditional Sections

Conditional sections are portions of the document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.

```
<!-- LaTeX math constructs -->
<!ENTITY % LaTeXmath  "INCLUDE">
<!ENTITY % latexmath.dtd SYSTEM "latexmath.dtd">
<![ %LaTeXmath; [
%latexmath.dtd;
]]>
<!ENTITY % MathML     "IGNORE">
<!ENTITY % latexmml.dtd SYSTEM "latexmml.dtd">
<![ %MathML; [ %latexmml.dtd; ]]>
```

8 Physical structures

- An XML document consists of one or more storage units (*entities*), that all have content and are identified by name.
- Each XML document has one entity called the *document entity*, which serves as the starting point for the XML processor and may contain the whole document.

- Entities may be either *parsed* or *unparsed*.
- A *parsed* entity's contents are referred to as its replacement text, that is considered an integral part of the document.
  Parsed entities are invoked by name using entity references.
- An *unparsed* entity is a resource whose contents may or may not be text, and if text, may not be XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.
  Unparsed entities are invoked by name, given in the value of `ENTITY` or `ENTITIES` attributes.
- *General entities* are entities for use within the document content.
- *Parameter entities* are parsed entities for use within the DTD.
- General and parameter entities use different forms of reference and are recognised in different contexts; they occupy different namespaces, so that a parameter entity and a general entity with the same name are two distinct entities.

8.1 Character references

A character reference refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.

- A character reference beginning with `&#x` corresponds to the hexadecimal representation of the character.
- A character reference beginning with `&#` corresponds to the decimal representation of the character.

Examples:

```
1:   &#xA;   <!-- A linefeed          -->
2:       <!-- Non-breaking space -->
3:   &#x420;   <!-- Russian r          -->
4:   &#x2207;  <!-- Nabla              -->
```

8.2 Entity references

- An entity reference refers to the content of a named entity.
- References to *parsed general entities* use ampersand (`&`) and semicolon (`;`) as delimiters.
- *Parameter-entity* references use percent-sign (`%`) and semicolon (`;`) as delimiters.

- In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with `standalone='yes'`, the `Name` given in the entity reference must match that in an entity declaration.
- In a well-formed document one needs not declare the following entities: `amp`, `lt`, `gt`, `apos`, `quot`, while in valid documents they *must* be declared in the DTD.
- The declaration of a parameter entity must precede any reference to it.
- The declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration.
- In a valid document with an external subset or external parameter entities with `standalone='no'`, the `Name` given in the entity reference must match that in an entity declaration.
- An entity reference must not contain the name of an unparsed entity.
- Unparsed entities may be referred to only in attribute values declared to be of type `ENTITY` or `ENTITIES`.
- A parsed entity must not contain a recursive reference to itself, either directly or indirectly.
- Parameter-entity references are only allowed in the DTD.

Example of character and entity references follow:

```
1:  The ampersand (&#38;) and less-than (&#x3e;) signs...
2:
```

```
 3:  This document was written on &date; by &Authors;.
 4:
 5:  <!-- declaration of the parameter entity "HTMLsymbol"... -->
 6:  <!ENTITY % HTMLsymbol PUBLIC
 7:          "-//W3C//ENTITIES Symbols//EN//HTML"
 8:          "http://www.w3.org/TR/xhtml1/DTD/HTMLsymbolx.ent">
 9:  <!-- ... and here it is referenced (in the DTD only!)    -->
10:  %HTMLsymbol;
```

8.3 Entity declarations

An internal entity is a parsed entity. An example follows:

```
<!ENTITY MathML "Mathematical Markup Language">
<!ENTITY XMLS   "&MathML; and other extensible markup languages">
```

If the entity is not internal, it is an external entity, declared using the following syntax.

```
1:   <!ENTITY myfile     SYSTEM "/user/goosssens/cern.xml''>
2:   <!ENTITY xhtml      PUBLIC
3:            "-//W3C//DTD XHTML 1.0 Transitional//EN"
4:            "/user/goossens/xml/dtds/transitional.dtd"
5:   <!ENTITY myFigure   SYSTEM "../cern.eps" NDATA eps>
```

8.4 Parsed entities

8.4.1 Text declaration

External parsed entities may each *begin* with a *text declaration.*

```
<?xml' version="1.0" encoding="..." ?>
```

- Each external parsed entity in an XML document may use a different encoding for its characters.
- All XML processors must be able to read entities in either UTF-8 or UTF-16.
- Entities encoded in ASCII (a subset of UTF-8) do not need an encoding declaration; entities containing texts using diacritics, encoded for instance in Latin 1 (`ISO-8859-1`), must *always* have their encoding specified.
- Possible other encoding are: `UTF-8`, `UTF-16`, `ISO-10646-UCS-2`, `ISO-10646-UCS-4` (corresponding to various transformations of Unicode/ISO/IEC 10646), `ISO-8859-1`, `ISO-8859-2`, etc. for the various parts of ISO-8859, `ISO-2022-JP`, `Shift_JIS`, `EUC-JP`, for various encoded forms of JIS X-0208-1997. For other encodings it is recommended to use character encodings registered (as charsets) with the Internet Assigned Numbers Authority [IANA]

```
1:  <?xml encoding='UTF-8'?><!-- default -->
2:  <?xml version="1.0" encoding="ISO-8859-1">
3:  <?xml encoding='EUC-JP'?>
```

8.5 Predefined Entities

All XML processors must recognise the five entities `amp`, `gt`, `lt`, `apos`, `quot`, whether they are declared or not

Valid documents must nevertheless declare these entities in their DTD if they are referenced.

```
1:   <!ENTITY lt     "&#38;#60;">
2:   <!ENTITY gt     "&#62;">
3:   <!ENTITY amp    "&#38;#38;">
4:   <!ENTITY apos   "&#39;">
5:   <!ENTITY quot   "&#34;">
```

The < and & characters in the declarations of `lt` and `amp` are doubly escaped to meet the requirement that entity replacement be well-formed.

9 A few simple examples

Consider a simple XML file `invitation1.xml`.

```
 1:  <?xml version="1.0"?>
 2:  <!DOCTYPE invitation SYSTEM "invitation1.dtd">
 3:  <invitation>
 4:  <!-- ++++ The header part of the document ++++ -->
 5:  <front>
 6:  <to>All HEP physicists and their friends</to>
 7:  <date>Tuesday 10 October to Thursday 12 October 2000</date>
 8:  <where>All over the LEP site</where>
 9:  <why>LEP Closure Fest</why>
10:  </front>
11:  <!-- +++++ The main part of the document +++++ -->
12:  <body>
13:  <par>
14:  To celebrate <emph>twelve good years</emph> of physics with LEP all
15:  CERN staff are invited to three days of festivities.
16:  </par>
17:  <par>
18:  Please do not miss this unique occasion to come and join us
19:  evening. And, do not forget to bring friends and colleagues.
20:  </par>
21:  <par>
22:  We <emph>really</emph> hope that you will be able to attend.
23:  </par>
24:  </body>
25:  <!-- +++ The closing part of the document ++++ -->
26:  <back>
27:  <signature>The LEP Gang</signature>
28:  </back>
29:  </invitation>
```

It has an associated DTD `invitation1.dtd`:

```
 1:  <!-- invitation DTD   -->
 2:  <!ELEMENT invitation (front, body, back) >
 3:  <!ELEMENT front      (to, date, where, why?) >
 4:  <!ELEMENT date       (#PCDATA) >
 5:  <!ELEMENT to         (#PCDATA) >
 6:  <!ELEMENT where      (#PCDATA) >
 7:  <!ELEMENT why        (#PCDATA) >
 8:  <!ELEMENT body       (par+) >
 9:  <!ELEMENT par        (#PCDATA|emph)* >
10:  <!ELEMENT emph       (#PCDATA) >
11:  <!ELEMENT back       (signature) >
12:  <!ELEMENT signature  (#PCDATA) >
```

We can check it with an validating XML parser, such as Apache's `xerces` [2]

```
> xerces -saxcount -v -w invitation1.xml
invitation1.xml: 100 ms (14 elems, 0 attrs, 18 spaces, 411 chars)
```

We can code the same information using attributes rather than elements. Here is the variant XML file `invitation2.xml`.

```
 1:  <?xml version="1.0"?>
 2:  <!DOCTYPE invitation SYSTEM "invitation2.dtd">
 3:  <invitation to="All HEP physicists and their friends"
 4:              date="Tuesday 10 October to Thursday 12 October 2000"
 5:              where="All over the LEP site"
 6:              why="LEP Closure Fest"
 7:              signature="The LEP Gang"
 8:  >
 9:  <par>
10:  To celebrate <emph>twelve good years</emph> of physics with LEP all
```

```
11:  CERN staff are invited to three days of festivities.
12:  </par>
13:  <par>
14:  Please do not miss this unique occasion to come and join us
15:  evening. And, do not forget to bring friends and colleagues.
16:  </par>
17:  <par>
18:  We <emph>really</emph> hope that you will be able to attend.
19:  </par>
20:  </invitation>
```

and its associated DTD `invitation2.dtd`.

```
 1:  <!-- invitation2 DTD     -->
 2:  <!ENTITY % i18n   "xml:lang NMTOKEN #IMPLIED">
 3:  <!ELEMENT invitation (par+)>
 4:  <!ATTLIST invitation
 5:                  %i18n;
 6:                  date      CDATA #REQUIRED
 7:                  to        CDATA #REQUIRED
 8:                  signature CDATA #REQUIRED
 9:                  where     CDATA #REQUIRED
10:                  why       CDATA #IMPLIED
11:  >
12:  <!ELEMENT par        (#PCDATA|emph)*>
13:  <!ELEMENT emph       (#PCDATA)>
```

Once more can check it with the parser:

```
> xerces -saxcount -v -w invitation2.xml
invitation2.xml: 116 ms (6 elems, 5 attrs, 4 spaces, 280 chars)
```

10 The Unicode character set

The Unicode standard ([24] and [11]) subdivides its character repertoire into a few classes: *base* characters (among others, these contain the alphabetic characters of the Latin alphabet, without diacritics), *ideographic* characters, and *combining* characters (among others, this class contains most diacritics). These classes combine to form the class of letters. Digits and extenders are also distinguished.

10.1 Unicode and ISO/IEC 10646-1

Unicode is closely aligned with the international standard ISO/IEC 10646-1, also known as UCS (*Universal Character Set*). By construction, Unicode is identical to ISO/IEC 10646-1 and its amendments.

The ISO/IEC 10646-1 standard has a much greater application area, since it is basically a 31-bit code; thus it can encode over two billion characters. The canonical form of ISO/IEC 10646-1 uses a four-dimensional coding space consisting of 256 three-dimensional *groups*. Each group consists of 256 two-dimensional *planes* with each plane containing 256 one-dimensional *rows*, each having 256 *cells*. Thus character codes consist of up to four octets, which, ordered from least to most significant, correspond to the cell (C-octet), row (R-octet), plane (P-octet), and group (G-octet) numbers.

The first plane (Plane 00) of Group 00 is called the *Basic Multilingual Plane* (BMP). It is, by construction, identical to the Unicode layout. The sixteen following planes (01 to 10 hex) are directly addressable with Unicode and cover abot one million character codes (planes 0f and 10 are in fact private planes reserved for user extensions).

The overview of the allocations in plane 0 for Unicode 3 is shown below.

```
0000 - 007F: Basic Latin
0080 - 00FF: Latin-1 Supplement
0100 - 017F: Latin Extended-A
0180 - 024F: Latin Extended-B
0250 - 02AF: IPA Extensions
```

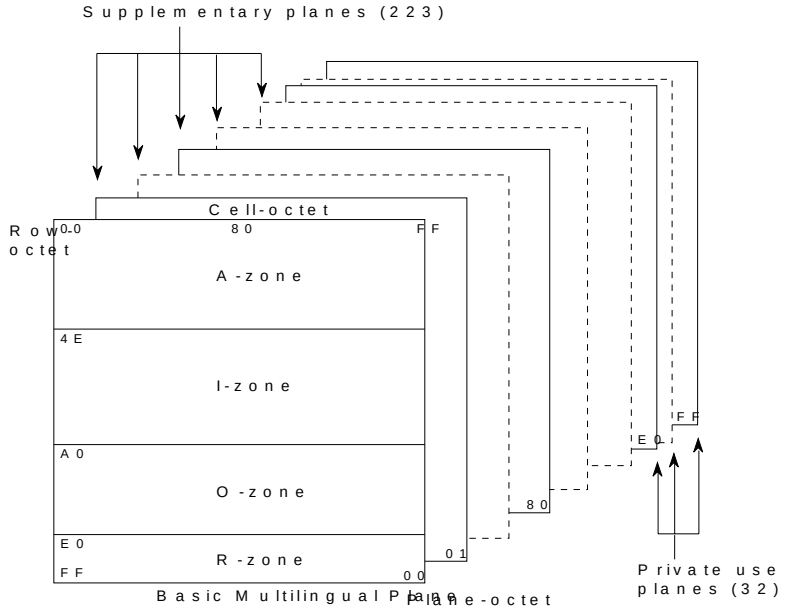

Figure 3: Structure of Group 00 of ISO/IEC 10646-1

02B0 - 02FF: Spacing Modifier Letters
0300 - 036F: Combining Diacritical Marks
0370 - 03FF: Greek
0400 - 04FF: Cyrillic
0530 - 058F: Armenian
0590 - 05FF: Hebrew
0600 - 06FF: Arabic
0700 - 074F: Syriac
0780 - 07BF: Thaana
0900 - 097F: Devanagari
0980 - 09FF: Bengali
0A00 - 0A7F: Gurmukhi
0A80 - 0AFF: Gujarati
0B00 - 0B7F: Oriya
0B80 - 0BFF: Tamil
0C00 - 0C7F: Telugu
0C80 - 0CFF: Kannada
0D00 - 0D7F: Malayalam
0D80 - 0DFF: Sinhala
0E00 - 0E7F: Thai
0E80 - 0EFF: Lao
0F00 - 0FFF: Tibetan
1000 - 109F: Myanmar
10A0 - 10FF: Georgian
1100 - 11FF: Hangul Jamo
1200 - 137F: Ethiopic
13A0 - 13FF: Cherokee
1400 - 167F: Unified Canadian Aboriginal Syllabics
1680 - 169F: Ogham
16A0 - 16FF: Runic
1780 - 17FF: Khmer
1800 - 18AF: Mongolian
1E00 - 1EFF: Latin Extended Additional
1F00 - 1FFF: Greek Extended
2000 - 206F: General Punctuation
2070 - 209F: Superscripts and Subscripts
20A0 - 20CF: Currency Symbols
20D0 - 20FF: Combining Marks for Symbols
2100 - 214F: Letterlike Symbols
2150 - 218F: Number Forms
2190 - 21FF: Arrows
2200 - 22FF: Mathematical Operators
2300 - 23FF: Miscellaneous Technical
2400 - 243F: Control Pictures
2440 - 245F: Optical Character Recognition
2460 - 24FF: Enclosed Alphanumerics

```
2500 - 257F: Box Drawing
2580 - 259F: Block Elements
25A0 - 25FF: Geometric Shapes
2600 - 26FF: Miscellaneous Symbols
2700 - 27BF: Dingbats
2800 - 28FF: Braille Patterns
2E80 - 2EFF: CJK Radicals Supplement
2F00 - 2FDF: Kangxi Radicals
2FF0 - 2FFF: Ideographic Description Characters
3000 - 303F: CJK Symbols and Punctuation
3040 - 309F: Hiragana
30A0 - 30FF: Katakana
3100 - 312F: Bopomofo
3130 - 318F: Hangul Compatibility Jamo
3190 - 319F: Kanbun
31A0 - 31BF: Bopomofo Extended
3200 - 32FF: Enclosed CJK Letters and Months
3300 - 33FF: CJK Compatibility
3400 - 4DB5: CJK Unified Ideographs Extension A
4E00 - 9FFF: CJK Unified Ideographs
A000 - A48F: Yi Syllables
A490 - A4CF: Yi Radicals
AC00 - D7A3: Hangul Syllables
D800 - DB7F: High Surrogates
DB80 - DBFF: High Private Use Surrogates
DC00 - DFFF: Low Surrogates
E000 - F8FF: Private Use
F900 - FAFF: CJK Compatibility Ideographs
FB00 - FB4F: Alphabetic Presentation Forms
FB50 - FDFF: Arabic Presentation Forms-A
FE20 - FE2F: Combining Half Marks
FE30 - FE4F: CJK Compatibility Forms
FE50 - FE6F: Small Form Variants
FE70 - FEFE: Arabic Presentation Forms-B
FEFF - FEFF: Specials
FF00 - FFEF: Halfwidth and Fullwidth Forms
FFF0 - FFFD: Specials
```

10.2 UTF-8 and UTF-16 encodings

The 32-bit ISO/IEC 10646-1 standard defines two transformation formats UTF-8 and UTF-16, which were also adopted by Unicode. The UTF-16 transformation assumes 16-bit characters, just like Unicode itself, but it allows for a certain range of characters to be used as an extension mechanism to access an additional million characters using 16-bit character pairs.

More useful is the UTF-8 transformation format that transforms all Unicode characters into a *variable length* encoding of up to three bytes. The first 128 characters (the ASCII subset) have the same bit sequences in UTF-8 and in ASCII, making it easy to handle documents using only that subset. This means that a lot of existing software can be used with Unicode (and UTF-8) without a major software rewrite.

More generally, UTF-8 can be described as a method to transform ISO 10646-1 streams into 8-bit streams, leaving ASCII as it is and expanding the other characters to up to six bytes. In fact, three bytes suffice for the 16-bit Unicode range; four bytes are enough for encoding about a million characters by reserving 2048 codepoints in Unicode (`D800` to `DFFF`) as an index. These one million codepoints should be enough for all the rare Chinese ideograms and historical scripts that do not fit into the Base Multilingual Plane of ISO 10646.

The relation between the 31 bit code ISO 10646 code and the up to six UTF-8 bytes is shown below. For Unicode, never more than three bytes are needed.

ISO 10646 range covered			UTF-8 representation
Bits	Hex Min	Hex Max	Byte Sequence in Binary
7	00000000	0000007f	0vvvvvvv
11	00000080	000007FF	110vvvvv 10vvvvvv
16	00000800	0000FFFF	1110vvvv 10vvvvvv 10vvvvvv
21	00010000	001FFFFF	11110vvv 10vvvvvv 10vvvvvv 10vvvvvv
26	00200000	03FFFFFF	111110vv 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv
31	04000000	7FFFFFFF	1111110v 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv

A UTF-8 octet that starts with binary 0 is a sequence of one (pure ASCII). Any octet starting with 10 is a trailing octet of a multioctet sequence. Any other octet is the start of a multioctet UTF-8 sequence, with the number of binary 1 digits indicating the number of octets of the multioctet encoding sequence. This makes it efficient to find the start of a character starting from an arbitrary location in an octet stream.

To allow coding for a million characters with UTF-16, the range 0000D800 to 0000DFFF is excluded from this conversion process. The following table shows in hexadecimal form the various ranges of the UTF-16 and UTF-8 sequences corresponding to the complete UCS-4 encoding. A semicolon shows the separation between the basic information unit in each case.

```
UCS-4           UTF-16        UTF-8

0000 0001;      0001;         01;
0000 007F;      007F;         7F;
0000 0080;      0080;         C2; 80;
0000 07FF;      07FF;         DF; BF;
0000 0800;      0800;         E0; A0; 80;
0000 FFFF;      FFFF;         EF; BF; BF;

0001 0000;      D800; DC00;   F0; 90; 80; 80;
0010 FFFF;      DBFF; DFFF;   F4; 8F; BF; BF;
001F FFFF;                    F7; BF; BF; BF;
0020 0000;                    F8; 88; 80; 80; 80;
03FF FFFF;                    FB; BF; BF; BF; BF;
0400 0000;                    FC; 84; 80; 80; 80; 80;
7FFF FFFF;                    FD; BF; BF; BF; BF; BF;
```

11 Non-Latin alphabetic languages

Since XML can handle Unicode in a native way, we expect it should not be too involved to code languages that use non-Latin alphabets, such as Russian, Greek, Chinese, Japanese, Arabic, and so on. In this section we show how you can handle Russian, Greek, and a little math in a single file without problems; the same applies to more complex situations, however.

The Yudit [22] Unicode editor supports input in several languages and allows onr to read and write in many encodings. It uses a relatively simple and intuitive graphical user interface.

Figure 4 shows an XML-coded file. Its first part shows three ways you can input Russian text. We reproduce (part) of these lines here for closer scrutiny:

```
 1:  <?xml version="1.0" encoding="UTF-8"?>
 2:  <!DOCTYPE  mydoc [
 3:  <!ELEMENT mydoc (#PCDATA)>
 4:  <!ENTITY % ISOcyr1 SYSTEM "ISOcyr1.pen">
 5:  %ISOcyr1;
 6:  ]>
 7:  <mydoc>
 8:  <par>The word Russian (Russkiĭ) in Cyrillic: <br/>
 9:  Using ISO Cyrillic set:
10:  &Rcy;&ucy;&scy;&scy;&kcy;&icy;&jcy;  <br/>
11:  Using XML Unicode entities:
12:  &#x0420;&#x0443;&#x0441;&#x0441;&#x043a;&#x0438;&#x0439;
13:  </par>
```

Lines 4–5 define an external entity set ISOcyr1, which will be read on the local system in the file ISOcyr1.pen. It contains definitions for Cyrillic letters, such as:

```
 1:  <!ENTITY rcy    "&#x440;"> <!--small er, Cyrillic -->
 2:  <!ENTITY Rcy    "&#x420;"> <!--capital ER, Cyrillic -->
 3:  <!ENTITY scy    "&#x441;"> <!--small es, Cyrillic -->
 4:  <!ENTITY Scy    "&#x421;"> <!--capital ES, Cyrillic -->
```

The above excerpt defines the small and capital Cyrillic letters *r* and *s* in function of their Unicode number. That is how we get Cyrillic letters by symbolic name on line 10 in our source file. Of course,

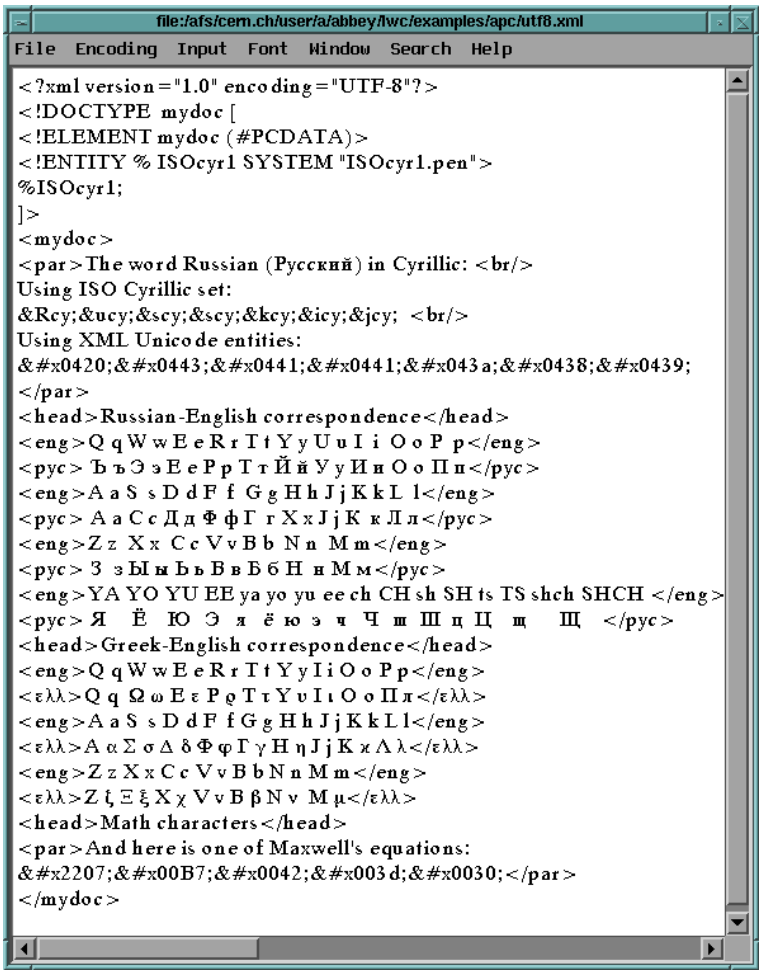

Figure 4: A UTF-8 encoded XML file with Russian, Greek, and math

as on line 12, we can also directly enter the Unicode character references ourselves (compare the entity references `&Rcy;` on line 10 and `&#x0420;` on line 12; decide which is more readable and maintainable). On the other hand, on line 8 we have entered Russian directly by using Yudit, which saved it as UTF-8, that is, two bytes for alphabetic non-ASCII characters. These byte-codes print funnily because they correspond to Unicode characters recoded as two-byte UTF-8 sequences (see previous section).

The final part of the XML file shows Unicode character references to write a small mathematics formula.

```
1:  <head>Math characters</head>
2:  <par>And here is one of Maxwell's equations:
3:  &#x2207;&#x00B7;&#x0042;&#x003d;&#x0030;</par>
4:  </mydoc>
```

12 XML namespaces

The aim of using *namespaces* is to guarantee uniqueness for the names of element types, attributes, etc., when the vocabularies of various languages (DTDs) are combined to reuse parts of already-written modules.

To solve this problem the W3C has published the recommendation *Namespaces in XML* on 14 January 1999 [32]

- an *XML namespace* is defined as a collection of names, identified by a URI reference;
- the use of the URI reference identifies uniquely element types and attribute names in an XML document;
- this URI is just a namespace identifier, and *does not* have to point to a real resource; it is merely a sequence of characters;
- the URI is used as a *prefix* to disambiguate element type and attribute names. A colon (`:`) is used to separate prefix and name.

12.1 Declaring namespaces

A namespace is declared using a family of reserved attributes. Such an attribute's name must either be `xmlns` or have `xmlns:` as a prefix. These attributes, like any other XML attributes, may be provided directly or by default.

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xt="http://www.jclark.com/xt"
```

12.2 Qualified names

Documents that conform to the XML namespace specification can use *qualified names*. They consist of a *prefix* that associates the name with a namespace URI reference in a namespace declaration and a *local part*. The prefix is only a placeholder for a namespace name and applications should use the namespace name, not the prefix, in constructing names whose scope extends beyond the containing document. The local part of the name should have a meaning inside the namespace corresponding to its prefix.

For instance, below the prefix `xsl:` is a placeholder for the string between double quotes on the first line, while `fo:` is a placeholder for the URI between double quotes on the second line. Each of the element type instances in the document using these prefixes will effectively be expanded so that the fully qualified name is used by the XML parser and down-stream applications.

```
1:  <xsl:stylesheet version="1.0"
2:                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3:                  xmlns:fo="http://www.w3.org/1999/XSL/Format">
4:  <xsl:template match="emph">
5:    <fo:inline-sequence font-style="italic">
6:      <xsl:apply-templates/>
7:    </fo:inline-sequence>
8:  </xsl:template>
```

12.3 Applying namespaces to elements and attributes

The namespace declaration is considered to apply to the element where it is specified and to all elements within the content of that element, unless overridden by another namespace declaration:

```
1:  <?xml version="1.0"?>
2:  <!-- both namespace prefixes are available throughout -->
3:  <bk:book xmlns:bk='urn:loc.gov:books'
4:           xmlns:isbn='urn:ISBN:0-395-36341-6'>
5:      <bk:title>Cheaper by the Dozen</bk:title>
6:      <isbn:number>1568491379</isbn:number>
7:  </bk:book>
```

An example of namespace scoping and using a default namespace is the following:

```
1:  <?xml version="1.0"?>
2:  <!-- unprefixed element have the HTML namespace -->
3:  <xsl:stylesheet version="1.0"
4:                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5:                  xmlns="http://www.w3.org/TR/xhtml1/strict">
6:    <xsl:template match="invitation/par">
7:    <p><xsl:apply-templates/></p>
8:  </xsl:template>
```

Namespaces can be nested, turned off locally, and be redefined locally.

```
<?xml version='1.0'?>
 ...
 <particles>
   <!-- HTML namespace -->
   <table xmlns='http://www.w3.org/TR/REC-html40'>
    <tr><td>Particle</td><td>Mass</td><td>Details</td></tr>
    <tr>
      <!-- no namespace for the following line -->
      <td xmlns=""><em>neutron</em></td>
      <td>939.56 MeV</td>
      <td>
        <pdg xmlns="otherNamespace">
         <!-- Here I locally use a new default namespace -->
         <quarks>udd</quarks><lifetime>886.7 s</lifetime>
         <decay>p,e,anue</decay>
        </pdg>
      </td><!-- we return to the HTML namespace -->
    </tr>
  </table>
 <!-- From here on we are back in the original namespace -->
 </particles>
 ...
```

13 Putting it all together

We are now ready to apply everything we have discussed to prepare a document using a *industry standard* DTD, especially optimized for technical document, namely DocBook [25]. We want to customise this DTD slighly, and, more importantly, include some mathematics markup tagged according to the MathML recommendation [31].

Let us consider the input XML source `dbmml.xml`, which follows:

```
<?xml version="1.0" ?>
<!DOCTYPE article SYSTEM
"/usr/local/share/docbookxml/3.17/docbookx.dtd" [
 <!ENTITY % local.bibliocomponent.mix "| note">
 <!ENTITY % equation.content "(math+)">
 <!ENTITY % inlineequation.content "(math+)">
 <!ENTITY % mathml SYSTEM "mathmldtd/mathml2.dtd">
 <!ATTLIST math xmlns CDATA #FIXED "http://www.w3.org/1998/Math/MathML" >
%mathml;
]>
<article>
<artheader>
<title>A Docbook document with a few formulae</title>
<author><firstname>Michel</firstname> <surname>Goossens</surname>
</author>
<pubdate>Wednesday May 24th 2000</pubdate>
<abstract>
<para>
This document is marked up according to the DocBook model. It also
includes a couple of simple formulae using MathML element tags.
</para>
</abstract>
</artheader>
<section>
<title>The DocBook model</title>
<para>
DocBook <xref role="bib" linkend="docbook" endterm="docbookab"/>
proposes an XML model
to markup technical documents, specially in the
areas of computer equipment or software.
</para>

```

```
<para>
The DocBook DTD contains hundreds of elements to clearly tag
the different elements of the source document (book, manual, article,
etc.), non only their hierarchical level but also their semantic
function.
The structure of the DTD has been optimised to allow customisation.
Thus, it is relatively easy to add or remove elements or attributes,
to change to content model of the containing elements appropriately,
or to limit the range of allowed values for attributes.
</para>
</section>

<section>
<title>MathML and mathematics</title>
<para>
MathML <xref role="bib" linkend="rec-mathml" endterm="rec-mathmlab"/>
is a W3C recommendation for coding mathematics.
</para>

<para>
MathML consists of the parts, presentation and content markup.
It permits the representation of long and complex equations.
MathML source code is easy to generate automatically and can be edited
without problems (even when the syntax seems verbose and complex at
first sight).
</para>
</section>

<section>
<title>A MathML example</title>
<para>
MathML can be typeset inline, as seen here
<inlineequation>
<math>
<mi>E</mi><mo>=</mo><mi>m</mi><msup><mi>c</mi><mn>2</mn></msup>
</math>
</inlineequation>, Einstein's famous formula.
</para>

<para>
An equation can also be highlighted by using a display with the help
of the element
<sgmltag class="element">informalequation</sgmltag>, as shown here for
a matrix:
</para>

<informalequation>
<math>
<mrow>
  <mi>A</mi>
  <mo>=</mo>
  <mfenced open="[" close="]">
    <mtable><!-- table or matrix   -->
      <mtr> <!-- row in a table    -->
         <mtd><mi>x</mi></mtd><!-- table -->
         <mtd><mi>y</mi></mtd><!-- entry -->
      </mtr>
      <mtr>
         <mtd><mi>z</mi></mtd>
         <mtd><mi>w</mi></mtd>
      </mtr>
    </mtable>
  </mfenced>
</mrow>
<mtext>.</mtext>
</math>
</informalequation>
```

```
100:  </section>
101:
102:  <bibliography>
103:  <title>References</title>
104:  <biblioentry id="docbook">
105:  <abbrev id="docbookab">WALSH99</abbrev>
106:  <authorgroup>
107:  <author><firstname>Norman</firstname><surname>Walsh</surname></author>
108:  <author><firstname>Leonard</firstname><surname>Muelner</surname></author>
109:  </authorgroup>
110:  <title>Docbook. The Definitive Guide.</title>
111:  <publisher>
112:  <publishername>O'Reilly &amp;amp; Associates, Inc.</publishername>
113:  </publisher>
114:  <copyright><year>1999</year></copyright>
115:  <isbn>1-56592-580-7</isbn>
116:  <note>
117:  <para>
118:  The DocBook reference Guide is at  <ulink
119:  url="http://www.oasis-open.org/docbook/"/>,
120:  DTDs and XSL stylesheets are at <ulink
121:  url="http://nwalsh.com/docbook/"/></para></note>
122:  </biblioentry>
123:  <biblioentry id="rec-mathml">
124:  <abbrev id="rec-mathmlab">MATHML99</abbrev>
125:  <authorgroup>
126:  <author><othername>World Wide Web Consortium</othername></author>
127:  <editor><firstname>Patrick</firstname><surname>Ion</surname></editor>
128:  <editor><firstname>Robert</firstname><surname>Miner</surname></editor>
129:  </authorgroup>
130:  <title>Mathematical Markup Language (MathML[tm]) 1.01 Specification
131:  <ulink url="http://www.w3.org/TR/REC-MathML/"/></title>
132:  </biblioentry>
133:  </bibliography>
134:  </article>
```

Let us study the first lines in more detail. Line 3 indicates where the DocBook DTD can be found on the system. With respect to that DTD on line 4 we add an element `node` into the bibliography (the DocBook DTD is specially structured to make this easy). Similarly we *replace* the content model for both the `equation` (line 5) and `inlineequation` (line 6) elements by one or more `math` elements, whose default namespace is that of the MathML language, namely `http://www.w3.org/1998/Math/MathML` (line 8). To allow MathML element to be used in the document, we define the whereabouts of (line 7) and then include (line 9) the MathML DTD. All elements in the document instance use the vocabulary defined by the DocBook DTD, except the elements inside the `math` elements (lines 67, 81-97), where we use the MathML (presentation markup) vocalulary.

We can typeset this document with, for instance, `xmltex` [4], and obtain the output shown in Figure 5

PartII
The Extensible Stylesheet language

14 Stylesheet languages: XSL

14.1 XSL: a little history

- Historically two approaches: CSS (HTML world) *against* DSSSL (computer science world).
- CSS: simple declarative scheme, started simple but is growing more complex (CSS1, CSS2, ...); targets essentially Web applications.
- DSSSL-o, a sub-set of DSSSL, was proposed in 1997 as a style language to transform XML documents, construct table of contents, indexes, complex page markup with generated text, etc., but this

A Docbook document with a few formulae

Michel GOOSSENS

Wednesday May 24th 2000

Abstract

This document is marked up according to the DocBook model. It also includes a couple of simple formulae using MathML element tags.

1 The DocBook model

DocBook [1] proposes an XML model to markup technical documents, specially in the areas of computer equipment or software.

The DocBook DTD contains hubdreds of elements to clearly tag the different elements of the source document (book, manual, article, etc.), non only their hierarchical level but also thier semantic function. The structure of the DTD has been optimized to allow customisation. Thus, it is relatively easy to add or remove elements or attributes, to change to content model of the containing elements appropriately, or to limit the range of allowed values for attributes.

2 MathML and mathematics

MathML [2] is a W3C recommendation for coding mathematics.

MathML consists of tho parts, presentation and content markup. It permits the representation of long and complex equations. MathML source code is easy to generate automatically and can be edited without problems (even when the syntax seems verbose and complex at first sight).

3 A MathML example

MathML can be typeset inline, as seen here $E = mc^2$, Einstein's famous formula.

An equation can also be highlighted by using a display with the help of the element `<informale-quation>`, as shown here for a matrix:

$$A = \left[\begin{array}{cc} x & y \\ z & w \end{array} \right].$$

References

[1] Norman WALSH Leonard MUELNER *Docbook. The Definitive Guide.* O'Reilly & Associates, Inc. 1999 1-56592-580-7

The DocBook reference Guide is at `http://www.oasis-open.org/docbook/`, DTDs and XSL stylesheets are at `http://nwalsh.com/docbook/`

[2] World Wide Web Consortium Patrick ION Robert MINER *Mathematical Markup Language (MathML[tm]) 1.01 Specification* `http://www.w3.org/TR/REC-MathML/`

Figure 5: DocBook document typeset with `xmltex`

was not accepted, amongst other considerations, because the Web community disliked the Scheme-like `(((...)))` syntax.

- A first proposal combining all CSS and DSSSL-o formatting objects (August 1997) using XML syntax was rejected as *too little too late*.
- To start the development of a an XML stylesheet language (XSL) the XSL WG was chartered on 23 January 1998.
- Successive working drafts were published in August and December 1998, then in April and July 1999. Each such draft was substantially different of its predecessor, making back-wards incompatible changes in the syntax (which is all right in principle, but annoying for early adopters).
- The current version has three parts: XPath [46] (addressing parts of an XML document), XSLT [48] (transforming XML documents into other XML documents), and XSLF [49] (formatting objects). The formatting objects part is presently nearing completion.

15 The XPath language

15.1 Principles

- XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations (XSLT) and XPointer.
- The primary purpose of XPath is to address parts of an XML document.
- XPath also provides basic facilities for manipulation of strings, numbers and booleans.

The primary syntactic construct in XPath is the expression. An expression matches the production Expr. An expression is evaluated to yield an object, which has one of the following four basic types:

- *node-set*, an unordered collection of nodes without duplicates;
- *boolean*, `true` or `false`;
- *number*, a floating-point number;
- *string*, a sequence of Unicode characters;

Expression evaluation occurs with respect to a context. XSLT and XPointer specify how the context is determined for XPath expressions used in XSLT and XPointer respectively. The context consists of:

- the context node;
- a pair of non-zero positive integers (the context position and the context size);
- a set of variable bindings;
- a function library;
- the set of namespace declarations in scope for the expression.

15.2 Location paths

Location paths, the most important construct in XPath, can be expressed using a straightforward but rather verbose syntax. There are also a number of syntactic abbreviations that allow common cases to be expressed concisely.

15.2.1 Location steps

A location step consists of three parts:

1. an *axis*, which specifies the tree relationship between the nodes selected by the location step and the context node;
2. a *node test*, which specifies the node type and expanded-name of the nodes selected by the location step;
3. zero or more *predicates*, which use arbitrary expressions to further refine the set of nodes selected by the location step.

```
axis-name :: node-test[predicate]*
```

15.2.2 Axes

The list of allowed axis types is the following:

```
'ancestor'    'ancestor-or-self'
'attribute'
'child'
'descendant'  'descendant-or-self'
'following'   'following-sibling'
```

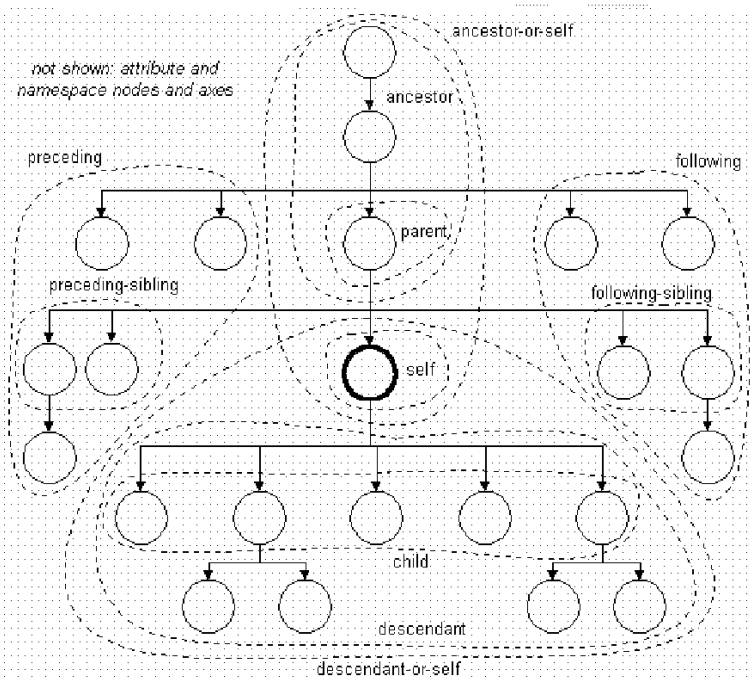

Figure 6: Visual representation of XPath axes

```
'namespace'
'parent'
'preceding'   'preceding-sibling'
'self'
```

15.3 Examples (full syntax)

- `child::para` selects the `para` element children of the context node
- `child::*` selects all element children of the context node
- `child::text()` selects all text node children of the context node
- `child::node()` selects all the children of the context node, whatever their node type
- `attribute::name` selects the `name` attribute of the context node
- `attribute::*` selects all the attributes of the context node
- `descendant::para` selects the `para` element descendants of the context node
- `ancestor::div` selects all `div` ancestors of the context node
- `ancestor-or-self::div` selects the `div` ancestors of the context node and, if the context node is a `div` element, the context node as well
- `descendant-or-self::para` selects the `para` element descendants of the context node and, if the context node is a `para` element, the context node as well
- `self::para` selects the context node if it is a `para` element, and otherwise selects nothing
- `child::chapter/descendant::para` selects the `para` element descendants of the `chapter` element children of the context node
- `child::*/child::para` selects all `para` grandchildren of the context node
- `/` selects the document root (which is always the parent of the document element)
- `/descendant::para` selects all the `para` elements in the same document as the context node
- `/descendant::olist/child::item` selects all the `item` elements in the same document as the context node that have an `olist` parent

- `child::para[position()=1]` selects the first `para` child of the context node
- `child::para[position()=last()]` selects the last `para` child of the context node
- `child::para[position()=last()-1]` selects the last but one `para` child of the context node
- `child::para[position()>1]` selects all the `para` children of the context node other than the first `para` child of the context node
- `following-sibling::chapter[position()=1]` selects the next `chapter` sibling of the context node
- `preceding-sibling::chapter[position()=1]` selects the previous `chapter` sibling of the context node
- `/descendant::figure[position()=42]` selects the forty-second `figure` element in the document
- `/child::doc/child::chapter[position()=5]/child::section[position()=2]` selects the second `section` of the fifth `chapter` of the `doc` document element
- `child::para[attribute::type="warning"]` selects all `para` children of the context node that have a `type` attribute with value `warning`
- `child::para[attribute::type='warning'][position()=5]` selects the fifth `para` child of the context node that has a `type` attribute with value `warning`
- `child::para[position()=5][attribute::type="warning"]` selects the fifth `para` child of the context node if that child has a `type` attribute with value `warning`
- `child::chapter[child::title='Introduction']` selects the `chapter` children of the context node that have one or more `title` children with `string-value` equal to `Introduction`
- `child::chapter[child::title]` selects the `chapter` children of the context node that have one or more `title` children
- `child::*[self::chapter or self::appendix]` selects the `chapter` and `appendix` children of the context node
- `child::*[self::chapter or self::appendix][position()=last()]` selects the last `chapter` or `appendix` child of the context node

15.4 Abbreviated syntax

There exists an abbreviated syntax for the most common XPath axes

- `div/para` is short for `child::div/child::para`
- `../title` is short for `parent::node()/child::title`
- `//para` is short for `/descendant-or-self::node()/child::para`
- `.//para` is short for `self::node()/descendant-or-self::node()/child::para`
- `para[@type="warning"]` is short for `child::para[attribute::type="warning"]`

Here are some examples of location paths using abbreviated syntax:

- `para` selects the `para` element children of the context node
- `*` selects all element children of the context node
- `text()` selects all text node children of the context node
- `@name` selects the `name` attribute of the context node
- `@*` selects all the attributes of the context node
- `para[1]` selects the first `para` child of the context node
- `para[last()]` selects the last `para` child of the context node
- `*/para` selects all `para` grandchildren of the context node
- `/doc/chapter[5]/section[2]` selects the second `section` of the fifth `chapter` of the `doc`
- `chapter//para` selects the `para` element descendants of the `chapter` element children of the context node
- `//para` selects all the `para` descendants of the document root and thus selects all `para` elements in the same document as the context node
- `//olist/item` selects all the `item` elements in the same document as the context node that have an `olist` parent

- `.` selects the context node
- `.//para` selects the `para` element descendants of the context node
- `..` selects the parent of the context node
- `../@lang` selects the `lang` attribute of the parent of the context node
- `para[@type="warning"]` selects all `para` children of the context node that have a `type` attribute with value `warning`
- `para[@type="warning"][5]` selects the fifth `para` child of the context node that has a `type` attribute with value `warning`
- `para[5][@type="warning"]` selects the fifth `para` child of the context node if that child has a `type` attribute with value `warning`
- `chapter[title="Introduction"]` selects the `chapter` children of the context node that have one or more `title` children with `string-value` equal to `Introduction`
- `chapter[title]` selects the `chapter` children of the context node that have one or more `title` children
- `employee[@secretary and @assistant]` selects all the `employee` children of the context node that have both a `secretary` attribute and an `assistant` attribute

15.5 Expressions

Parentheses may be used for grouping.

15.5.1 Function calls

An argument is converted to type `string` as if by calling the `string` function. An argument is converted to type `number` as if by calling the `number` function. An argument is converted to type `boolean` as if by calling the `boolean` function. An argument that is not of type `node-set` cannot be converted to a `node-set`. It is an error if the number or type of arguments is wrong.

15.5.2 Booleans

The precedence order is (from least important to most important):

1. `or`
2. `and`
3. `=`, `!=`
4. `<=`, `<`, `=>=>`

The operators are all left associative. For example, `3 > 2 > 1` is equivalent to `(3 > 2) > 1`, which evaluates to `false`.

Note that the < character must be escaped as `<`. However, it might be easier to invert the inequality and use > instead.

16 The XSLT transformation language

16.1 Introductory remarks

XSLT transforms an XML input document into a hierarchical result tree, which often is serialised as an XML instance (see Figure 7).

You can also serialise the output as XSL formatting objects, which can be further transformed into a *printable* or *viewable* output format (see Figure 8)

Finally, you can also generate non-XML output, that can be interpreted by a dedicated parser (I shall give an example of direct transformation into LaTeX).

The XSLT processor ignored the DTD and any comments and processing instructions present in the input XML document.

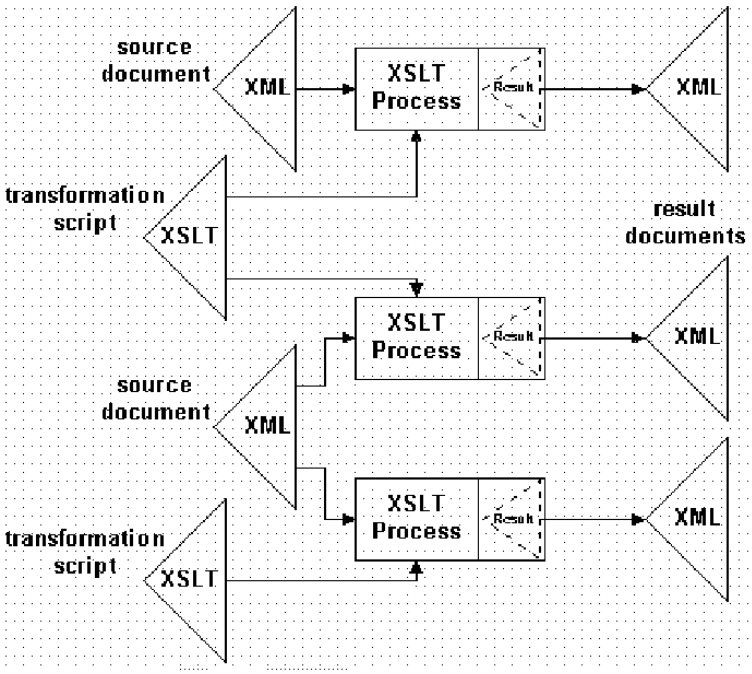

Figure 7: XSLT transformation of the XML document tree

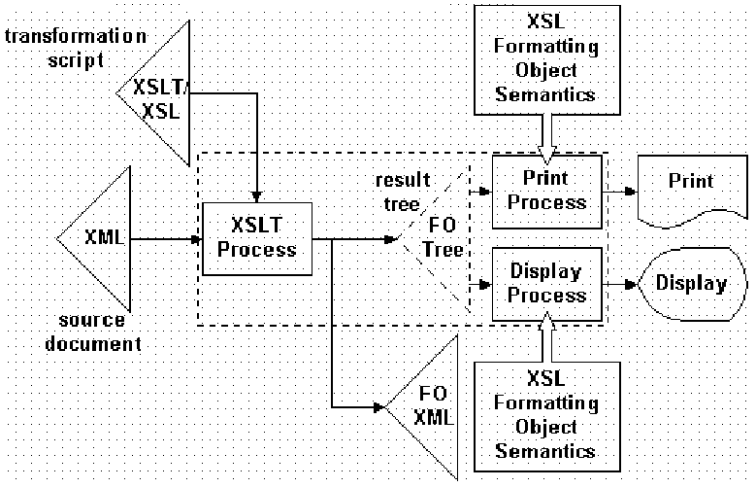

Figure 8: Creating XSL formatting objects

The output result tree can be constructed by *pulling* the data (mainly using <xsl:value-of> and <xsl:for-each> instructions). This approach worked all right if the structure of the input source is quite static and known.

Alternatively, when the source structure is only partly known it might be better to *push* the input data, i.e., let the construction of the result tree be driven by the data themselves. Here one can use the <apply-templates> instruction.

In most practical situations a combination of both these approaches is probably a winning strategy.

16.2 A simple example

Let us start with the simplest possible stylesheet, an empty one.

```
1:  <xsl:stylesheet version="1.0"
2:     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3:  </xsl:stylesheet>
```

When we run this stylesheet with one of the XSL processors (`xt` or `lotusxsl`), we get the following

```
1:  > saxonxsl welcome.xml empty.xsl
2:  <?xml version="1.0" encoding="utf-8" ?>Welcome to CERN in Geneva!
3:  > oraxsl  welcome.xml empty.xsl
4:  <?xml version = '1.0' encoding = 'UTF-8'?>
5:  Welcome to CERN in Geneva!
```

So we see that the input is just copied through to the output (some precessors, e.g., the Microsoft XSL processors, will show nothing since they require that you add a specific template rule for copying input to output.)

We now construct a stylesheet which is a little more complicated for our example:

```
1:  <?xml version='1.0'?>
2:  <xsl:stylesheet version='1.0'
3:         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:
5:  <xsl:output method="html"/>
6:
7:  <xsl:template match="/">
8:  <html>
9:  <em><b><xsl:apply-templates/></b></em>
10: </html>
11: </xsl:template>
12: </xsl:stylesheet>
```

When we run that with `xt` we get

```
1:  > xtxsl welcome.xml welcome.xsl
2:  <html>
3:  <em><b>Welcome to CERN in Geneva!</b></em>
4:  </html>
```

as we would expect. If we would not have used the `<xsl:apply-templates/>` instruction, the data would not have been copied through to the output!

In the above example the data were pushed through to the output, but we could also pull them ourselves, although this example is too trivial to make the point clearly. The file `welcome1.xsl` follows.

```
1:  <?xml version='1.0'?>
2:  <xsl:stylesheet version='1.0'
3:          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:
5:  <xsl:output method="html"/>
6:
7:  <xsl:template match="/">
8:  <html>
9:  <em><b><xsl:value-of select="."/></b></em>
10: </html>
11: </xsl:template>
12: </xsl:stylesheet>
```

As it is so common that XML files will be transformed into HTML, a special simplified syntax, without explicit templates, is provided (in fact the template for the root node is implicit).

```
1:  <html  xsl:version='1.0'
2:        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3:        xmlns="http://www.w3.org/TR/xhtml1">
4:  <em><b><xsl:value-of select="."/></b></em>
5:  </html>
```

which gives the following output

```
1:  > xtxsl welcome.xml welcomehtml.xsl
2:  <?xml version="1.0" encoding="utf-8"?>
3:  <html xmlns="http://www.w3.org/TR/xhtml1">
4:  <em><b>Welcome to CERN in Geneva!</b></em>
5:  </html>
```

16.3 The structure of an XSL stylesheet

The various elements that are possible at the top level of a stylesheet are listed below.

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/XSL/Transform">

  <xsl:import href="..."/>
  <xsl:include href="..."/>
  <xsl:strip-space elements="..."/>
  <xsl:preserve-space elements="..."/>
  <xsl:output method="..."/>
  <xsl:key name="..." match="..." use="..."/>
  <xsl:decimal-format name="..."/>
  <xsl:namespace-alias stylesheet-prefix="..." result-prefix="..."/>

  <xsl:attribute-set name="...">     ...    </xsl:attribute-set>
  <xsl:variable name="...">          ...    </xsl:variable>
  <xsl:param name="...">             ...    </xsl:param>
  <xsl:template match="...">         ...    </xsl:template>
  <xsl:template name="...">          ...    </xsl:template>

</xsl:stylesheet>
```

16.4 Template rules

16.4.1 How an XSLT processor works

- A XSLT transformation (an XSL stylesheet) describes rules for transforming a *source* tree into a *result* tree.
- The transformation is achieved by associating patterns with templates.
- A pattern is matched against elements in the source tree.
- A template is instantiated to create part of the result tree, which is separate from the source tree, and whose structure can be completely different from the structure of the source tree.
- In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.
- When a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates. Instructions can select and process descendant source elements.
- Note that elements are only processed when they have been selected by the execution of an instruction.
- The result tree is constructed by finding the template rule for the root node and instantiating its template.
- When finding the applicable template rule, more than one template rule may have a pattern that matches a given element. However, only one template rule will be applied. A conflict resolution mechanism exists.
- XSLT makes use of the XPath expression language for selecting elements for processing, for conditional processing and for generating text.
- *Extension mechanisms* are defined for extending the set of instruction elements used in templates and for extending the set of functions used in XPath expressions.

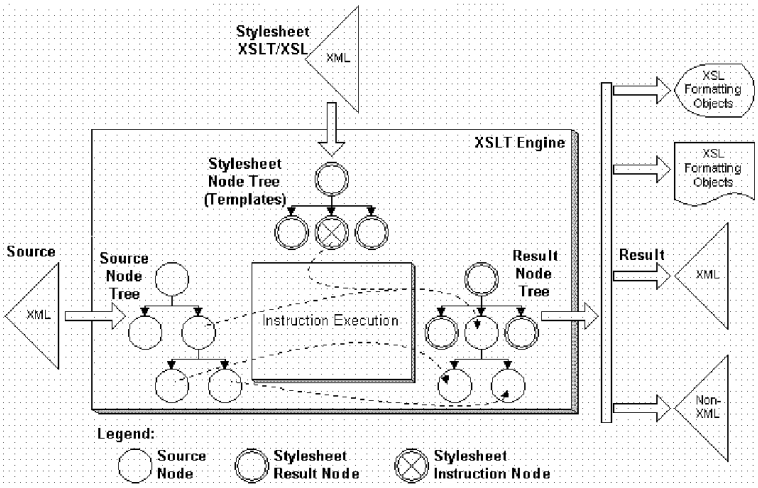

Figure 9: XSL at work

Figure 9 shows graphically how a single instruction in the stylesheet obtains information about a particular point in the source tree. Part of the input tree and then copied or transformed and copied to the result tree.

16.4.2 Patterns

- Template rules identify the nodes to which they apply by using a pattern.
- Patterns are also used for numbering and for declaring keys.
- A pattern specifies a set of conditions on a node.
- A node that satisfies the conditions matches the pattern; a node that does not satisfy the conditions does not match the pattern.
- The syntax for patterns is a subset of the syntax for XPath expressions.

A few examples of patterns follow.

- `chapter|appendix` matches any `chapter` element and any `appendix` element;
- `aaa/bbb` matches any `bbb` element with an `aaa` parent;
- `aaa//bbb` matches any `bbb` element with an `aaa` ancestor;
- `/` matches the root node, `text()` matches any text node, `processing-instruction()` matches any processing instruction, `node()` matches any node other than an attribute node and the root node;
- `id("CERN-IT-ASD")` matches the element with unique ID `CERN-IT-ASD`;
- `appendix//list/item[position()>1]` matches any `item` element that has a `list` parent that has an `appendix` ancestor, and that is not the first `item` child of its parent;
- `bbb[position() mod 2 = 1]` is true for any `bbb` element that is an odd-numbered `bbb` child of its parent.
- `chapter[@class="appendix"]//para` matches any `para` element with a `chapter` ancestor element that has a class attribute with value `appendix`.
- `@class` matches any class attribute (not any element that has a class attribute), and `@*` matches any attribute.

16.4.3 Defining template rules

A template rule is specified with the `xsl:template` element. The `match` attribute is a `Pattern` that identifies the source node or nodes to which the rule applies.

The following template rule matches `hi` elements and has a template, which produces a `fo:inline-sequence` formatting object with a `font-weight` property of `italic`.

```
1:  <xsl:template match="hi">
2:    <fo:inline-sequence font-style="italic">
3:      <xsl:apply-templates/>
4:    </fo:inline-sequence>
5:  </xsl:template>
```

The `<xsl:apply-templates/>` instruction processes all of the children of the current node, including text nodes.

Conflict resolution is handled *explicitly* (with a `priority` attribute, or *implicitly* via a set of rules (which behave *as one would expect* in most cases). When in doubt, use the `priority` attribute.

16.4.4 Modes and built-in templates

Modes allow an element to be processed multiple times, each time producing a different result. Such a mode can be specified on both `xsl:template` and `xsl:apply-templates` instruction via an optional `mode` attribute.

There is a built-in template rule to allow recursive processing to continue in the absence of a successful pattern match by an explicit template rule in the stylesheet. This template rule applies to both element nodes and the root node.

```
1:  <xsl:template match="*|/">
2:    <xsl:apply-templates/>
3:  </xsl:template>
```

There is a built-in template rule for each mode, which allows recursive processing to continue in the same mode in the absence of a successful pattern match by an explicit template rule in the stylesheet.

```
1:  <xsl:template match="*|/" mode="m">
2:    <xsl:apply-templates mode="m"/>
3:  </xsl:template>
```

Similarly, the built-in template rule for text and attribute nodes just copies text through:

```
1:  <xsl:template match="text()|@*">
2:    <xsl:value-of select="."/>
3:  </xsl:template>
```

The built-in template rule for processing instructions and comments is to do nothing.

```
1:  <xsl:template match="processing-instruction()|comment()"/>
```

These built-in template rules have have a lower import precedence than all other template rules. Thus, the author can override a built-in template rule by including an explicit template rule.

17 Invitation examples

17.1 LaTeX output

We first can transform the document `invitation1.xml` into LaTeX. For this we consider the XSL stylesheet `invlat1.xsl`:

```
 1:  <?xml version='1.0'?>
 2:  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 3:
 4:  <xsl:output method="text" indent="no" encoding="ISO-8859-1"/>
 5:
 6:  <xsl:strip-space elements="*"/>
 7:
 8:  <xsl:template match="invitation">
 9:  <xsl:text>\documentclass[12pt]{article}
10:  \usepackage{invitation}
11:  \begin{document}
12:  </xsl:text>
13:  <xsl:apply-templates/>
```

```
14:  <xsl:text>\end{document}
15:  </xsl:text>
16:  </xsl:template>
17:
18:  <xsl:template match="front">
19:  <xsl:text>\begin{Front}
20:  \To{</xsl:text>
21:  <xsl:value-of select="to"/>
22:  <xsl:text>}
23:  \Date{</xsl:text>
24:  <xsl:value-of select="date"/>
25:  <xsl:text>}
26:  \Where{</xsl:text>
27:  <xsl:value-of select="where"/>
28:  <xsl:text>}
29:  \Why{</xsl:text>
30:  <xsl:value-of select="why"/>
31:  <xsl:text>}
32:  \end{Front}
33:  </xsl:text>
34:  </xsl:template>
35:
36:  <xsl:template match="body">
37:  <xsl:text>\begin{Body}
38:  </xsl:text>
39:    <xsl:apply-templates/>
40:  <xsl:text>
41:  \end{Body}
42:  </xsl:text>
43:  </xsl:template>
44:
45:  <xsl:template match="par">
46:  <xsl:text>
47:  \par </xsl:text>
48:  <xsl:apply-templates/>
49:  </xsl:template>
50:
51:  <xsl:template match="emph">
52:  <xsl:text>\emph{</xsl:text>
53:  <xsl:apply-templates/>
54:  <xsl:text>}</xsl:text>
55:  </xsl:template>
56:
57:  <xsl:template match="back">
58:  <xsl:text>\begin{Back}
59:  \Signature{</xsl:text>
60:  <xsl:value-of select="signature"/>
61:  <xsl:text>}
62:  \end{Back}
63:  </xsl:text>
64:  </xsl:template>
65:
66:  </xsl:stylesheet>
```

In the stylesheet we have defined for each element in the input source the equivalent LaTeX command mostly inside `<xsl:text>` elements, in particular for the root element `invitation` (lines 9-11) we write the LaTeX preamble, while on line 14 we close the document gracefully. Line 13 with the XSL instruction `<xsl:apply-templates/>` instructs the XSL processor to look inside the `invitation` element to continue processing the enclosed elements.

The XSL transformation associates a specific LaTeX command with each of the XML element, which is translated into basic LaTeX command with the help of the package `invitation.sty` (loaded on line 10). These generic command names used allow us to support multiple languages, if needed.

```
 1:  % invitation.sty
 2:  % Package to format invitation.xml
 3:  \setlength{\parskip}{1ex}
 4:  \setlength{\parindent}{0pt}
 5:  \pagestyle{empty}%% Turn off page numbering
 6:  \RequirePackage{array}
 7:  \newenvironment{Front}%
 8:    {\begin{center}\huge \sffamily Memorandum\end{center}
 9:     \begin{flushleft}
10:     \begin{tabular}{@{}>{\bfseries}p{.2\linewidth}@{}p{.8\linewidth}@{}}\hline
11:    }
12:    {To whom:  & \@To     \\
13:     Occasion: & \@Why    \\
14:     Venue:    & \@Where  \\
```

Memorandum

To whom:	All HEP physicists and their friends
Occasion:	LEP Closure Fest
Venue:	All over the LEP site
When:	Tuesday 10 October to Thursday 12 October 2000

To celebrate *twelve good years* of physics with LEP all CERN staff are invited to three days of festivities.

Please do not miss this unique occasion to come and join us. And, do not forget to bring friends and colleagues.

We *really* hope that you will be able to attend.

The LEP Gang

Figure 10: Result of formatting the XML document with LaTeX

```
15:     When:     & \@Date   \\\hline
16:     \end{tabular}
17:     \end{flushleft}
18:    }
19:  \newenvironment{Body}{\vspace*{\parskip}}{\vspace*{\parskip}}
20:  \newenvironment{Back}
21:    {\begin{flushleft}}
22:    {\hspace*{.5\linewidth}\fbox{\emph{\@Sig}}
23:     \end{flushleft}
24:    }
25:  \newcommand{\To}[1]{\gdef\@To{#1}}
26:  \newcommand{\Date}[1]{\gdef\@Date{#1}}
27:  \newcommand{\Where}[1]{\gdef\@Where{#1}}
28:  \newcommand{\Why}[1]{\gdef\@Why{#1}}
29:  \newcommand{\Signature}[1]{\gdef\@Sig{#1}}
30:  \endinput
```

With an XSL processor (James Clark's `xt` in this case) we can obtain a TeX file that can then be printed:

```
java com.jclark.xsl.sax.Driver invitation1.xml invlat1.xsl invlat1.tex
```

The result (after using LaTeX and `dvips` is shown in Figure 10.

17.2 HTML output

Another important application of XSLT transformation is preparing HTML pages. To show how this works we start from the XML file `>invitation2.xml`, which only contains three element, of which the root element has several attributes. Globally this file conatins the same information que le fichier `invitation1.xml`.

To transform the XML file `invitation2.xml` into HTML we use the XSL stylesheet `invhtml2.xsl` which follows:

```
1:  <?xml version='1.0'?>
2:  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3:
4:  <xsl:output method="html"/>
5:  <xsl:preserve-space elements="*"/>
6:
7:  <xsl:template match="invitation">
8:  <html>
9:  <head>
10:  <title> Invitation (XSL/CSS formatting) </title>
11:  <link href="invit.css" rel="stylesheet" type="text/css"/>
12:  <!-- 20 May 2000 mg -->
```

```
13:  </head>
14:  <body>
15:  <h1>INVITATION</h1>
16:  <table>
17:  <tbody>
18:  <tr><td class="front">To: </td>
19:  <td><xsl:value-of select="@to"/></td></tr>
20:  <tr><td class="front">When: </td>
21:  <td><xsl:value-of select="@date"/></td></tr>
22:  <tr><td class="front">Venue: </td>
23:  <td><xsl:value-of select="@where"/></td></tr>
24:  <tr><td class="front">Occasion: </td>
25:  <td><xsl:value-of select="@why"/></td></tr>
26:  </tbody>
27:  </table>
28:  <xsl:apply-templates/>
29:  <p class="signature"><xsl:value-of select="@signature"/></p>
30:  </body>
31:  </html>
32:  </xsl:template>
33:
34:  <xsl:template match="par">
35:  <p><xsl:apply-templates/></p>
36:  </xsl:template>
37:
38:  <xsl:template match="emph">
39:  <em><xsl:apply-templates/></em>
40:  </xsl:template>
41:
42:  </xsl:stylesheet>
```

Line 4 defines the output format as HTML, so that we can use HTML directly inside the templates. Lines 7, 34 and 42 handle the three elements used in our document. The information for the date, subject, etc. are extracted from the attributes of the `invitation` elemnt with the help of the `<xsl:value-of .../>` instructions (lines 19, 21, etc.). The title is generated with an `h1` (line 15) element, while the front matter is generated with the help of a table (lines 16 to 27). For certain elements we have specified a `class` attribute, so that we can control their presentation with the help of a CSS stylesheet [27]. Line 11 associates the CSS stylesheet `invit.css` with the HTML file. The contents of this CSS file follows:

```
 1:  /* CSS stylesheet for invitation1 in HTML */
 2:  BODY {margin-top: 1em;     /* global page parameters */
 3:        margin-bottom: 1em;
 4:        margin-left: 1em;
 5:        margin-right: 1em;
 6:        font-family: serif;
 7:        line-height: 1.1;
 8:        color: black;
 9:  }
10:  H1   {text-align: center;  /* for global title   */
11:        font-size: x-large;
12:  }
13:  P    {text-align: justify; /* paragraphs in body */
14:        margin-top: 1em;
15:  }
16:  TABLE { border-width: 0pt }
17:  TBODY { border-width: 0pt }
18:  TD[class="front"] {        /* table data in front matter */
19:        text-align: left;
20:        font-weight: bold;
21:  }
22:  TD.front {        /* table data in front matter */
23:        text-align: left;
24:        font-weight: bold;
25:  }
26:  EM   {font-style: italic;  /* emphasis in body   */
27:  }
28:  P.signature {     /* signature          */
29:        text-align: right;
30:        font-weight: bold;
31:  }
```

We obtain the result HTML file `invhtml2.html` with the help of an XSL processor, such as `xt` as follows:

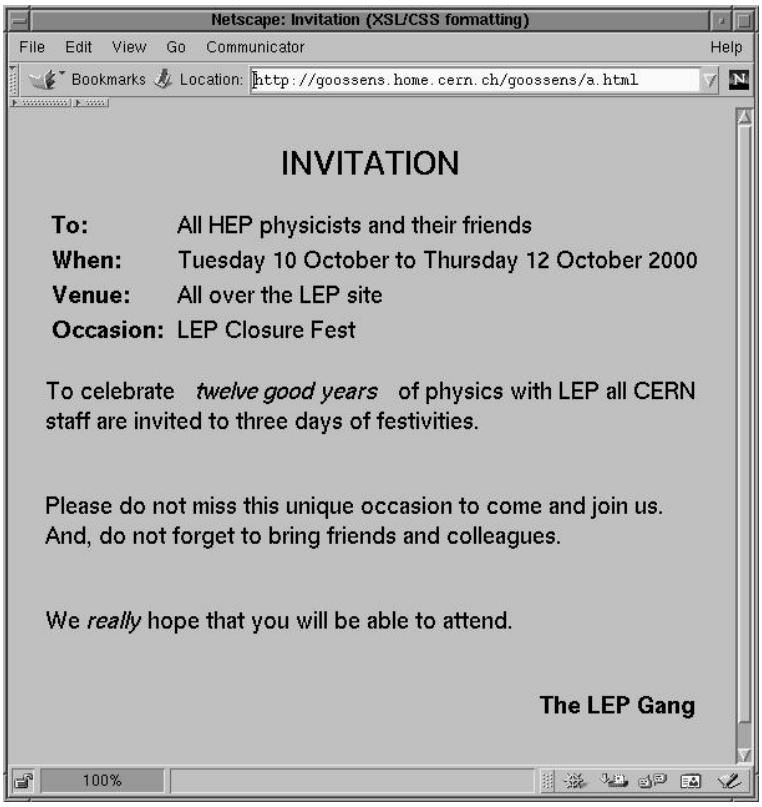

Figure 11: The HTML file `invhtml2.html` viewed with Netscape and an associated CSS style

```
java com.jclark.xsl.sax.Driver invitation2.xml invhtml2.xsl invhtml2.html
```

Figure 11 shows the resulting HTML file as viewed with the Netscape 4.07 browser together with the the CSS stylesheet `invit.css`

17.3 XSL formatting objects

The XSL stylesheet `invfo1.xsl` transforms the XML file `invitation1.xml` file into XSL formatting objects.

```
 1:
 2:  <?xml version='1.0'?>
 3:  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 4:                               xmlns:fo="http://www.w3.org/1999/XSL/Format">
 5:
 6:  <xsl:strip-space elements="*"/>
 7:
 8:  <!-- Parameterizations -->
 9:
10:  <xsl:variable name="PageMarginTop">75pt</xsl:variable>
11:  <xsl:variable name="PageMarginBottom">125pt</xsl:variable>
12:  <xsl:variable name="PageMarginLeft">80pt</xsl:variable>
13:  <xsl:variable name="PageMarginRight">150pt</xsl:variable>
14:  <xsl:variable name="BodyFont">Times-Roman</xsl:variable>
15:  <xsl:variable name="BodySize">12pt</xsl:variable>
16:  <xsl:variable name="TypeWriterFont">Computer-Modern-Typewriter</xsl:variable>
17:  <xsl:variable name="SansFont">Helvetica</xsl:variable>
18:  <xsl:variable name="ListRightMargin">12pt</xsl:variable>
19:  <xsl:variable name="ListAbove">12pt</xsl:variable>
20:  <xsl:variable name="ListBelow">12pt</xsl:variable>
21:  <xsl:variable name="ListNormalIndent">15pt</xsl:variable>
22:  <xsl:variable name="BulletOne">&#x2022;</xsl:variable>
23:
```

```
<xsl:template name="listitem">
  <xsl:param name="labeltext">labeltext</xsl:param>
  <xsl:param name="itemid">itemid</xsl:param>
  <xsl:param name="itemtext">itemtext</xsl:param>
  <fo:list-item id="{$itemid}">
    <fo:list-item-label color="green"  font-style="italic">
      <fo:block>
        <xsl:value-of select="$labeltext"/>
        <xsl:text>:</xsl:text>
      </fo:block>
    </fo:list-item-label>
    <fo:list-item-body>
      <fo:block><xsl:value-of select="$itemtext"/></fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>

<xsl:template match='/'>
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master
        page-master-name="allpages"
        margin-top="{$PageMarginTop}"
        margin-bottom="{$PageMarginBottom}"
        margin-left="{$PageMarginLeft}"
        margin-right="{$PageMarginRight}">
        <fo:region-body margin-bottom="100pt"/>
        <fo:region-after extent="25pt"/>
      </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence flow-name="allpages">
        <fo:flow font-family="serif">
          <xsl:apply-templates/>
        </fo:flow>
      </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="invitation/front">
<fo:block font-family="sans-serif"  font-size="24pt" color="blue"
          font-weight="bold"       text-align="center"
          space-after.optimum="24pt">
  <xsl:text>INVITATION</xsl:text>
</fo:block>

<fo:list-block provisional-distance-between-starts="2cm"
               provisional-label-separation="6pt">
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">To</xsl:with-param>
    <xsl:with-param name="itemid">listto</xsl:with-param>
    <xsl:with-param name="itemtext"><xsl:value-of select="to"/></xsl:with-param>
  </xsl:call-template>
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">When</xsl:with-param>
    <xsl:with-param name="itemid">listdate</xsl:with-param>
    <xsl:with-param name="itemtext"><xsl:value-of select="date"/></xsl:with-param>
  </xsl:call-template>
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">Venue</xsl:with-param>
    <xsl:with-param name="itemid">listwhere</xsl:with-param>
    <xsl:with-param name="itemtext" select="where"/>
  </xsl:call-template>
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">Occasion</xsl:with-param>
    <xsl:with-param name="itemid">listwhy</xsl:with-param>
    <xsl:with-param name="itemtext"><xsl:value-of select="why"/></xsl:with-param>
  </xsl:call-template>
</fo:list-block>
</xsl:template>

<xsl:template match="invitation/body/par">
  <fo:block space-before.optimum="{$BodySize}">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="invitation/body/par/emph">
  <fo:inline font-style="italic">
    <xsl:apply-templates/>
  </fo:inline>
```

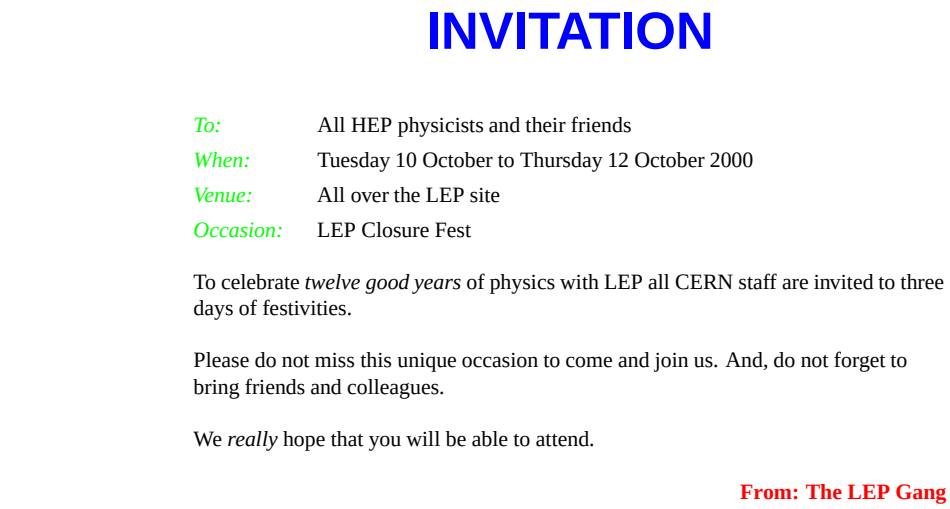

Figure 12: Rendering XSL formatting objects with PassiveTEX

```
104:  </xsl:template>
105:
106:  <xsl:template match="invitation/back">
107:  <fo:block color="red" space-before.optimum="{$BodySize}"
108:            font-weight="bold" text-align="end">
109:     <xsl:text>From: </xsl:text>
110:     <xsl:value-of select="signature"/>
111:  </fo:block>
112:  </xsl:template>
113:
114:  </xsl:stylesheet>
```

We do not want to go into the details of XSL formatting object (see [49] for more details). We merely mention line 4 that defines the XSL FO namespace. Note also how XSL variables are defined on lines 10-14 and used, for example, on lines 38-41. The `listitem` named template is defined on lines 16-31 with its three parameters, whose default values can be changed when the template is instantiated (lines 68-72, 73-77, 78-82 and 83-87).

Using the `xt` XSL processor we obtain the XML file `invfo1.fo`.

```
> java com.jclark.xsl.sax.Driver invitation.xml invfo1.xsl invfo1.fo
```

The file `invfo1.fo` must be formatted by complementary applications. Here we show the output obtained with PassiveTEX [20]. PassiveTEX uses `xmltex` [4] and pdfTEX (or TEX) to generate PDF or PostScript output. (Figure 12).

18 XSLT as a transformation language for manipulating XML data

The use of XSLT as database manupulation and query language is being debated since a long time, and efforts to define an XML query language, directly based on XSLT or not, are contining. In fact, following the W3C Query Languages Workshop in December 1998 [33], the mission of the XML Query working group is to provide flexible query facilities to extract data from real and virtual documents on the Web. The XML Query working group has published its Requirements Document [45] and the underlying data model [44]

18.1 Cleaning up the HTML source

To show the principle of how to use XSLT as transformation language we downloaded a series of HTML tables from a European Union Web page [8]. They contain information about the world's currencies and countries in ten languages. We shall be working with the French and English versions. As the table was split over two files and contained quite complex HTML4 markup, we first concatenated the tables and passed then through `tidy` [19] to get clean XML files. We also eliminated embelishments, notes, and other data not directly related to countries and their currencies. A stylesheet `filter.xsl` that gets rid of redundant markup follows.

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="ISO-8859-1"/>
<xsl:strip-space elements="td"/>

<xsl:template match="*" priority="-2">
<xsl:copy>
<xsl:apply-templates/>
</xsl:copy>
</xsl:template>

<xsl:template match="sup|a|i">
<!-- ignorer contenu -->
</xsl:template>
<xsl:template match="small|b">
<xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

The resulting file `frisotab-in.xml` looks like what follows:

```
<html>
<head>
<title>Pays et Monnaies</title>
</head>
<body>
<table>
<tr>
<td>Code ISO</td>
<td>Forme courte</td>
<td>Forme longue</td>
<td>Capitale</td>
<td>Citoyen ou habitant</td>
<td>Adjectif</td>
<td>Unité monétaire</td>
<td>Code ISO</td>
<td>Subdivision de l'unité monétaire</td>
</tr>

<tr>
<td>AD</td>
<td>Andorre</td>
<td>la Principauté d'Andorre</td>
<td>Andorre-la-Vieille</td>
<td>Andorran</td>
<td>andorran</td>
<td>peseta; franc français</td>
<td>ESP; FRF</td>
<td>centimo; centime</td>
</tr>
```

18.2 Prepare the database

The stylesheet `isotab1.xsl` will translate the row entries of the HTML table into a series of individual elements with a name indicating their content.

```
<?xml version='1.0' ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" encoding="ISO-8859-1"/>
<xsl:strip-space elements="*"/>

```

```
 7:  <xsl:template match="/html">
 8:  <xsl:element name="countries">
 9:  <xsl:text>
10:  </xsl:text>
11:  <xsl:apply-templates/>
12:  </xsl:element>
13:  </xsl:template>
14:
15:  <xsl:template match="*" priority="-2">
16:  <xsl:element name="UNKNOWN/INCONNU">
17:  <xsl:value-of select="name(.)"/>
18:  <xsl:apply-templates/>
19:  </xsl:element>
20:  </xsl:template>
21:
22:  <xsl:template match="head|tr">
23:  <!-- ignore content -->
24:  </xsl:template>
25:
26:  <xsl:template match="body|table">
27:  <xsl:apply-templates/>
28:  </xsl:template>
29:
30:  <xsl:template match="tr[position()>1]" priority="2">
31:  <xsl:element name="country">
32:  <xsl:text>&#xA;</xsl:text>
33:  <xsl:apply-templates/>
34:  </xsl:element>
35:  <xsl:text>&#xA;</xsl:text>
36:  </xsl:template>
37:
38:  <xsl:template match="td[1]">
39:  <xsl:element name="shortname">
40:  <xsl:apply-templates/>
41:  </xsl:element><xsl:text>&#xA;</xsl:text>
42:  </xsl:template>
43:
44:  <xsl:template match="td[2]">
45:  <xsl:element name="fullname">
46:  <xsl:apply-templates/>
47:  </xsl:element><xsl:text>&#xA;</xsl:text>
48:  </xsl:template>
49:
50:  <xsl:template match="td[3]">
51:  <xsl:element name="isocountry">
52:  <xsl:apply-templates/>
53:  </xsl:element><xsl:text>&#xA;</xsl:text>
54:  </xsl:template>
55:
56:  <xsl:template match="td[4]">
57:  <xsl:element name="capital">
58:  <xsl:apply-templates/>
59:  </xsl:element><xsl:text>&#xA;</xsl:text>
60:  </xsl:template>
61:
62:  <xsl:template match="td[5]">
63:  <xsl:element name="citizen">
64:  <xsl:apply-templates/>
65:  </xsl:element><xsl:text>&#xA;</xsl:text>
66:  </xsl:template>
67:
68:  <xsl:template match="td[6]">
69:  <xsl:element name="adjective">
70:  <xsl:apply-templates/>
71:  </xsl:element><xsl:text>&#xA;</xsl:text>
72:  </xsl:template>
73:
74:  <xsl:template match="td[7]">
75:  <xsl:element name="currency">
76:  <xsl:apply-templates/>
77:  </xsl:element><xsl:text>&#xA;</xsl:text>
78:  </xsl:template>
79:
80:  <xsl:template match="td[8]">
81:  <xsl:element name="isocurrency">
82:  <xsl:apply-templates/>
83:  </xsl:element><xsl:text>&#xA;</xsl:text>
84:  </xsl:template>
85:
86:  <xsl:template match="td[9]">
```

```
87:  <xsl:element name="currensubunit">
88:  <xsl:apply-templates/>
89:  </xsl:element><xsl:text>&#xA;</xsl:text>
90:  </xsl:template>
91:
92:  </xsl:stylesheet>
```

An XSL processor will transform the original XML file, as follows:

```
> oraxsl frisotab-in.xml isotab1.xsl > frisotab1.xml
```

The beginning of the file `frisotab1.xml` is as follows:

```
 1:  <countries>
 2:   <country>
 3:    <shortname>AD</shortname>
 4:    <fullname>Andorre</fullname>
 5:    <isocountry>la Principauté d'Andorre</isocountry>
 6:    <capital>Andorre-la-Vieille</capital>
 7:    <citizen>Andorran</citizen>
 8:    <adjective>andorran</adjective>
 9:    <currency>peseta; franc français</currency>
10:    <isocurrency>ESP; FRF</isocurrency>
11:    <currensubunit>centimo; centime</currensubunit>
12:   </country>
```

We can obtain the same output file from the source data by *pulling* the element content from the input by using <`xsl:value-of .../`> instructions, rather that *pushing* the data through the XSL filter and let the input drive the output by using <`apply-templates/`> matching patterns. Here is the alternate XSL stylesheet `isotab1-bis.xsl`:

```
 1:  <?xml version='1.0' encoding="ISO-8859-1"?>
 2:  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 3:  <xsl:output method="xml" encoding="ISO-8859-1"/>
 4:  <xsl:strip-space elements="*"/>
 5:
 6:  <xsl:template match="/html">
 7:  <xsl:element name="countries">
 8:  <xsl:text>
 9:  </xsl:text>
10:  <xsl:apply-templates/>
11:  </xsl:element>
12:  </xsl:template>
13:
14:  <xsl:template match="*" priority="-2">
15:  <xsl:element name="UNKNOWN/INCONNU">
16:  <xsl:apply-templates/>
17:  </xsl:element>
18:  </xsl:template>
19:
20:  <xsl:template match="head|tr" priority="0">
21:  <!-- ignore content -->
22:  </xsl:template>
23:
24:  <xsl:template match="body|table">
25:  <xsl:apply-templates/>
26:  </xsl:template>
27:
28:  <xsl:template match="table/tr[position()>1]" priority="2">
29:  <xsl:element name="country">
30:  <xsl:text>
31:  </xsl:text>
32:  <xsl:element name="shortname"><xsl:value-of select="td[1]"/></xsl:element>
33:  <xsl:element name="fullname"><xsl:value-of select="td[2]"/></xsl:element>
34:  <xsl:element name="isocountry"><xsl:value-of select="td[3]"/></xsl:element>
35:  <xsl:element name="capital"><xsl:value-of select="td[4]"/></xsl:element>
36:  <xsl:element name="citizen"><xsl:value-of select="td[5]"/></xsl:element>
37:  <xsl:element name="adjective"><xsl:value-of select="td[6]"/></xsl:element>
38:  <xsl:element name="currency"><xsl:value-of select="td[7]"/></xsl:element>
39:  <xsl:element name="isocurrency"><xsl:value-of select="td[8]"/></xsl:element>
40:  <xsl:element name="currensubunit"><xsl:value-of select="td[9]"/></xsl:element>
41:  <xsl:text>
42:  </xsl:text>
43:  </xsl:element>
```

```
<xsl:text>
</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

18.3 A database using attributes rather than elements

Rather than store the nine basic information items inside elements, we can store them inside attributes of a single `country` element. The following XSL stylesheet `isotab2.xsl`, which is quite similar to `isotab1-bis.xsl` shown previously, implements this:

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="ISO-8859-1"/>
<xsl:strip-space elements="*"/>

<xsl:template match="/html">
<xsl:element name="countries">
<xsl:text>
</xsl:text>
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="*" priority="-2">
<xsl:element name="UNKNOWN/INCONNU">
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="head|tr" priority="0">
<!-- ignore content -->
</xsl:template>

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

<xsl:template match="table/tr[position()>1]" priority="2">
<xsl:element name="country">
<xsl:attribute name="shortname"><xsl:value-of select="td[1]"/></xsl:attribute>
<xsl:attribute name="fullname"><xsl:value-of select="td[2]"/></xsl:attribute>
<xsl:attribute name="isocountry"><xsl:value-of select="td[3]"/></xsl:attribute>
<xsl:attribute name="capital"><xsl:value-of select="td[4]"/></xsl:attribute>
<xsl:attribute name="citizen"><xsl:value-of select="td[5]"/></xsl:attribute>
<xsl:attribute name="adjective"><xsl:value-of select="td[6]"/></xsl:attribute>
<xsl:attribute name="currency"><xsl:value-of select="td[7]"/></xsl:attribute>
<xsl:attribute name="isocurrency"><xsl:value-of select="td[8]"/></xsl:attribute>
<xsl:attribute name="currensubunit"><xsl:value-of select="td[9]"/></xsl:attribute>
</xsl:element>
<xsl:text>
</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

The resulting XML file begins like this:

```
<countries>
<country shortname="AD"
        fullname="Andorre"
        isocountry="la Principauté d'Andorre"
        capital="Andorre-la-Vieille"
        citizen="Andorran"
        adjective="andorran"
        currency="peseta; franc français"
        isocurrency="ESP; FRF"
        currensubunit="centimo; centime"/>
</countries>
```

We can transform the element-based representation into the attribute-based representation or the reverse with the help of the stylesheets `isotab1to2` (elements to attributes) and `isotab2to1` (attributes to elements). Both of them follow.

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<!-- isotab1to2.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" encoding="ISO-8859-1"/>

<xsl:template match="/">
<xsl:element name="countries"><xsl:text>&#xA;</xsl:text>
 <xsl:for-each select="countries/country">
  <xsl:element name="country">
   <xsl:attribute name="shortname"><xsl:value-of select="shortname"/></xsl:attribute>
   <xsl:attribute name="fullname"><xsl:value-of select="fullname"/></xsl:attribute>
   <xsl:attribute name="isocountry"><xsl:value-of select="isocountry"/></xsl:attribute>
   <xsl:attribute name="capital"><xsl:value-of select="capital"/></xsl:attribute>
   <xsl:attribute name="citizen"><xsl:value-of select="citizen"/></xsl:attribute>
   <xsl:attribute name="adjective"><xsl:value-of select="adjective"/></xsl:attribute>
   <xsl:attribute name="currency"><xsl:value-of select="currency"/></xsl:attribute>
   <xsl:attribute name="isocurrency"><xsl:value-of select="isocurrency"/></xsl:attribute>
   <xsl:attribute name="currensubunit"><xsl:value-of select="currensubunit"/></xsl:attribute>
  </xsl:element><xsl:text>&#xA;</xsl:text>
 </xsl:for-each>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

and

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<!-- isotab2to1.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" encoding="ISO-8859-1"/>

<xsl:template match="/">
<xsl:element name="countries"><xsl:text>&#xA;</xsl:text>
 <xsl:for-each select="countries/country">
  <xsl:element name="country"><xsl:text>&#xA;</xsl:text>
   <xsl:apply-templates select="{.}"/>
   <xsl:element name="shortname"><xsl:value-of select="@shortname"/></xsl:element>
   <xsl:element name="fullname"><xsl:value-of select="@fullname"/></xsl:element>
   <xsl:element name="isocountry"><xsl:value-of select="@isocountry"/></xsl:element>
   <xsl:element name="capital"><xsl:value-of select="@capital"/></xsl:element>
   <xsl:element name="citizen"><xsl:value-of select="@citizen"/></xsl:element>
   <xsl:element name="adjective"><xsl:value-of select="@adjective"/></xsl:element>
   <xsl:element name="currency"><xsl:value-of select="@currency"/></xsl:element>
   <xsl:element name="isocurrency"><xsl:value-of select="@isocurrency"/></xsl:element>
   <xsl:element name="currensubunit"><xsl:value-of select="@currensubunit"/></xsl:element>
   <xsl:text>&#x0A;</xsl:text>
  </xsl:element><xsl:text>&#x0A;</xsl:text>
 </xsl:for-each>
</xsl:element><xsl:text>&#x0A;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Note the high degree of symmetry between these two stylesheets. We have introduced some linefeeds `<xsl:text>
</xsl:text>` in appropriate places to make the resulting XML file more readable.

To fully benefit from the possibilities of XML we can also translate all element and attribute names into the native language of the user, in this case French, with the following stylesheet `isotab1to1fr.xsl`:

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" encoding="ISO-8859-1"/>

<xsl:template match="/">
<xsl:element name="listeDesPays"><xsl:text>&#x0A;</xsl:text>
 <xsl:for-each select="countries/country">
  <xsl:element name="pays"><xsl:text>&#x0A;</xsl:text>
   <xsl:element name="codeISO"><xsl:value-of select="shortname"/></xsl:element><xsl:text>&#x0A;</xsl:text>
   <xsl:element name="nomComplet"><xsl:value-of select="fullname"/></xsl:element><xsl:text>&#x0A;</xsl:text>
   <xsl:element name="nomISO"><xsl:value-of select="isocountry"/></xsl:element><xsl:text>&#x0A;</xsl:text>
   <xsl:element name="capitale"><xsl:value-of select="capital"/></xsl:element><xsl:text>&#x0A;</xsl:text>
   <xsl:element name="citoyen"><xsl:value-of select="citizen"/></xsl:element><xsl:text>&#x0A;</xsl:text>
   <xsl:element name="adjectif"><xsl:value-of select="adjective"/></xsl:element><xsl:text>&#x0A;</xsl:text>
   <xsl:element name="monnaie"><xsl:value-of select="currency"/></xsl:element><xsl:text>&#x0A;</xsl:text>
```

```
17:     <xsl:element name="monnaieCodeISO"><xsl:value-of select="isocurrency"/></xsl:element><xsl:text>&#x0A;</xsl:text>
18:     <xsl:element name="monnaieSousunité"><xsl:value-of select="currensubunit"/></xsl:element><xsl:text>&#x0A;</xsl:text>
19:    </xsl:element><xsl:text>&#x0A;</xsl:text>
20:   </xsl:for-each>
21:  </xsl:element><xsl:text>&#x0A;</xsl:text>
22:  </xsl:template>
23:
24:  </xsl:stylesheet>
```

We introduced again linefeeds to increase readability, and used *Camel* notation for the element (and attribute) names. The start of the generated file is as follows:

```
 1:  <listeDesPays>
 2:  <pays>
 3:  <codeISO>AD</codeISO>
 4:  <nomComplet>Andorre</nomComplet>
 5:  <nomISO>la Principauté d'Andorre</nomISO>
 6:  <capitale>Andorre-la-Vieille</capitale>
 7:  <citoyen>Andorran</citoyen>
 8:  <adjectif>andorran</adjectif>
 9:  <monnaie>peseta; franc français</monnaie>
10:  <monnaieCodeISO>ESP; FRF</monnaieCodeISO>
11:  <monnaieSousInité>centimo; centime</monnaieSousunité>
12:  </pays>
```

18.4 Using the database

Now that we have our data in a convenient form we want to use our database, for instance we want to obtain a list if countries with their name (simple and ISO) and their capital. We use the French version of the database and the XSL stylesheet `frisotab1exa1.xsl` that follows:

```
 1:  <?xml version='1.0' encoding="ISO-8859-1"?>
 2:  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 3:
 4:  <xsl:output method="text" encoding="ISO-8859-1"/>
 5:
 6:  <xsl:template match="/listeDesPays">
 7:  <xsl:for-each select="pays">
 8:  <xsl:value-of select="nomComplet"/>
 9:  <xsl:text> (</xsl:text>
10:  <xsl:value-of select="nomISO"/>
11:  <xsl:text>) et la capitale </xsl:text>
12:  <xsl:value-of select="capitale"/>
13:  <xsl:text>.&#x0A;</xsl:text><!-- linefeed -->
14:  </xsl:for-each>
15:  </xsl:template>
16:
17:  </xsl:stylesheet>
```

The resulting file is as follows:

```
1:  Andorre (la Principauté d'Andorre) et la capitale Andorre-la-Vieille.
2:  les Émirats arabes unis (les Émirats arabes unis) et la capitale Abou Dhabi.
3:  l'Afghanistan (l'É islamique d'Afghanistan) et la capitale Kaboul.
4:  Antigua-et-Barbuda (Antigua-et-Barbuda) et la capitale Saint John's.
```

We can also generate an HTML (or LaTeX) table from our data, showing a lsit of currencies sorted alphabetically, with its currency sub-unit, its ISO code, and the country where the currency is used. We also use the second form (using attributes) to generate the table with the help of the XSL stylesheet `frisotab2exa3.xsl`, which follows:

```
1:  <?xml version="1.0" encoding="ISO-8859-1"?>
2:  <html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
3:  <body bgcolor="white">
4:  <h1>Les monnaies et les pays</h1>
5:  <table border="1" rules="rows">
6:  <thead>
7:   <tr>
8:    <td align="center"><b>monnaie</b></td>
```

Les monnaies et les pays

monnaie	sous-unité	code ISO	pays
afghani	pul	AFA	l'Afghanistan
baht	satang	THB	la Thaïlande
balboa	centesimo	PAB	le Panama
birr éthiopien	cent	ETB	l'Érythrée
birr éthiopien	cent	ETB	l'Éthiopie
bolivar	centavo	VEB	le Venezuela
boliviano	centavo	BOB	la Bolivie
cedi	pesewa	GHC	le Ghana
centavo			le Nicaragua
colon du Costa Rica	centimo	CRC	Costa Rica
colon du Salvador	centavo	SVC	le Salvador; l'El Salvador
couronne danoise	øre	DKK	le Danemark
couronne danoise	øre	DKK	les Iles Féroé
couronne danoise	øre	DKK	le Groenland
couronne estonienne	sent	EEK	l'Estonie
couronne islandaise	eyrir	ISK	l'Islande
couronne norvégienne	øre	NOK	la Norvège
couronne norvégienne	øre	NOK	les Iles Svalbard et Jan Mayen
couronne slovaque	halier	SKK	la Slovaquie
couronne suédoise	öre	SEK	la Suède
couronne tchèque	halér	CZK	la République tchèque

Figure 13: A table with currencies and countries

```
 9:    <td align="center"><b>sous-unité</b></td>
10:    <td align="center"><b>code ISO</b></td>
11:    <td align="center"><b>pays</b></td>
12:   </tr>
13:  </thead>
14:  <xsl:for-each select="/listeDesPays/pays">
15:  <xsl:sort select="@monnaie"/>
16:   <xsl:if test="string(@monnaie) != '-'">
17:    <tr>
18:     <td><xsl:value-of select="@monnaie"/></td>
19:     <td><xsl:value-of select="@monnaieSousunité"/></td>
20:     <td align="center"><tt><xsl:value-of select="@monnaieCodeISO"/></tt></td>
21:     <td><xsl:value-of select="@nomComplet"/></td>
22:    </tr>
23:   </xsl:if>
24:  </xsl:for-each>
25:  </table>
26:  </body>
27:  </html>
```

Figure (see Figure 13) shows the resulting table viewed with Netscape.

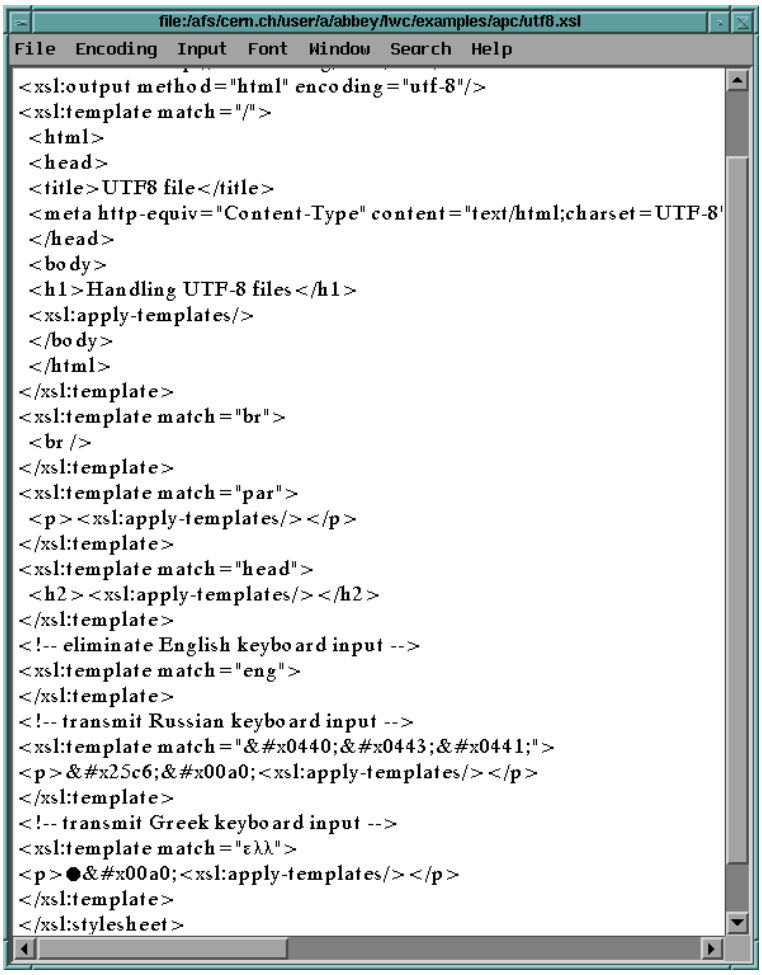

Figure 14: XSL file to transform UTF-8 encoded XML file with Russian, Greek, and math into HTML

19 Dealing with Non-Latin languages

Our next task is to transform this file onto something we can browse or print. The style sheet `utf8.xsl` (Figure 14) transforms the earlier XML file `utf8.xml` into HTML.

The most interesting part of the style sheet is its beginning. The line `<xsl:out` defines the output format as HTML and the output encoding as UTF-8. This allows us to output straight HTML tags as seen in the template for the root element `/`, where we enclose the whole output file inside a correct HTML structure. The meaning of the remaining lines should be straightforward. The empty template for the `eng` template eliminates those elements. Because XSL style sheets are genuine XML files, we can use the full set of alphabetic and ideographic Unicode characters; in particular we can refer to the non-Latin element names of our XML document (see the last match pattern for the Greek in the stylesheet). In fact, we can mix native as well as character reference codes in the XSL file to refer to Unicode characters. Note the use of the nonbreaking space (` `) between the diamond or bullet and the subsequent text.

We can run these two files through an XSL parser, such as `xt`, and output the HTML file `utf8.html`

```
> xt utf8.xml utf8.xsl utf8.html
```

The result as viewed with Netscape is shown in Figure 15

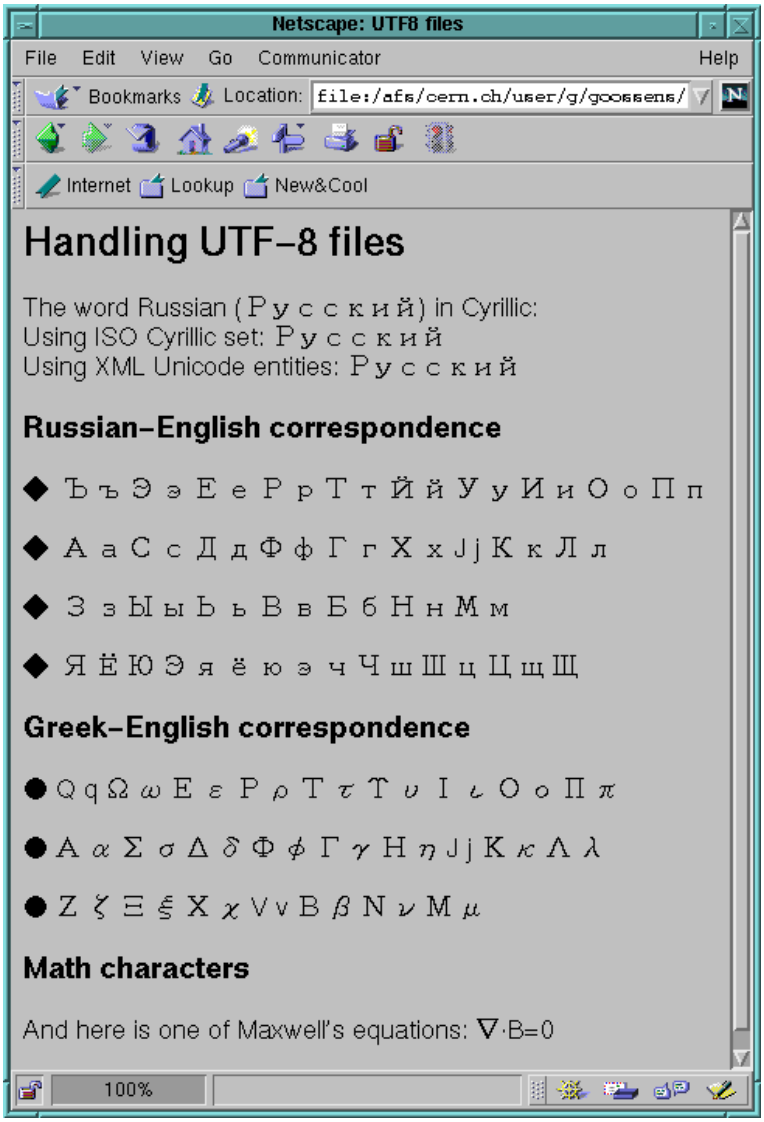

Figure 15: HTML rendering of UTF8 file with Netscape

PartIII
XML-based specifications

20 Xpointer

20.1 Goals and principles

The Abstract of the current working draft [47] says:

... the XML Pointer Language (XPointer) [is] the language to be used as a fragment identifier for any URI-reference [15] that locates a resource of Internet media type `text/xml` *or* `application/xml`, where these media types are defined in [26].

XPointer builds on XPath, the XSLT expression language. It extends XPath to express any user selection on the document in the following areas:

- addresses points and ranges as well as nodes;
- can locate information by string matching;
- can use addressing expressions in URI-references as fragment identifiers.

20.2 Fragment identifiers

XPointer fragment identifiers allow one full form of XPointer addressing and two shorthand forms.

20.2.1 Full XPointers

The full form of addressing consists of one or more fragment parts. Each starts with a scheme name and is followed by a parenthesized expression. When the scheme is `xpointer`, the associated expression provides XPath-compatible access to nodes in an XML document's information set, and adds access to any selection of an XML document's content. Such locations are specified using a variety of descriptive techniques.

20.2.2 Bare Names

The bare name form of addressing is provided for HTML compatibility. The appearance of the name stands for the same name provided as the argument of a location step using the `id()` function, e.g., the following URI-reference fragment identifier:

```
intro
```

is the bare-name equivalent of the following full-form fragment identifier:

```
xpointer(id("intro"))
```

This reserved shorthand should encourage use of IDs, which are the form of addressing most likely to survive document change. At the same time it provides an analog of the HTML fragment identifier behavior.

20.2.3 Child Sequences

The child sequence form of addressing locates an element by stepwise navigation using a sequence of integers separated by slashes (/). Each integer `n` locates the `n`th child element of the previously located element, equivalent to an XPath location step of the form `*[n]`. Nodes other than elements cannot be located by child sequences.

Navigation begins at an element located either by specifying a name, or by the string `/1`, which locates the document element. For example, the following two XPointers would locate the same XML information, where intro is the ID attribute value of an element that is the fifth child of the second child of the document element:

```
intro/14/3
/1/2/5/14/3
```

20.3 Main XPointer extensions to XPath

- A generalization of the XPath concept of *nodes* to the XPointer concept of *locations*, which subsumes nodes, points, and ranges.
- Two new location types, `point` and `range`, that can appear in location-set results as well as tests for these location types.

- The function `string-range()`, which uses the range location type for selections that do not correspond to single XML information set nodes.
- The functions `here()` and `origin()`, to provide for addressing relative to the location of an XPointer expression itself, and to the point of origin for hypertext traversal.
- The functions `start-point()` and `end-point()`, to address the beginning and ending locations which bound another location such as a node or range.
- The predicate function `unique()`, to enable testing whether an XPointer expression or sub-expression locates a single location rather than multiple locations or no locations.

20.4 Examples

Select everything from the beginning of the last `P` descendant element in a section with ID `sec2.1` through the end of the first `P` descendant element in another section with ID `sec2.2`.

```
1:  xpointer(id("sec2.1")/descendant::P[last()]/range-to(id("sec2.2")/descendant::P[last()]))
```

Select everything starting at the last but on paragraph (`p[last()-1]`) of the element with the identifier `section3.1` up to the second paragraph (`p[2]`) of the element with identifier `section3.3`.

```
1:  xpointer(id('section3.1')/p[last()-1]/range-to(id('section3.3')/p[2]))
```

The following XPointer locates the range from the element with `ID "chap1"` to the element with `ID "chap2"`.

```
1:  xpointer(id("chap1")/range-to(id("chap2")))
```

In a document that uses the empty elements <`REVST/`> for revision start and <`REVEND/`> for revision end to mark the boundaries of edits, the following XPointer would select, for each revision, a range starting at the beginning of the `REVST` element and ending at the end of the next `REVEND` element.

```
1:  xpointer(descendant::REVST/range-to(following::REVEND[1]))
```

Return a range that selects the 17th occurrence of the string Thomas Pynchon occurring in a `title` element.

```
1:  string-range(//title,"Thomas Pynchon")[17]
```

Here are two ways to return a collapsed range whose points immediately precede the letter P (8 from the start of the string) in the third occurrence of the string Thomas Pynchon in a `P` element.

```
1:  string-range(//P,"Thomas Pynchon",8,0)[3]
2:  string-range(string-range(//P,"Thomas Pynchon")[3],"P",1,0)
```

Select the fifth exclamation mark in any text node in the document and the character immediately following it.

```
1:  string-range(/,"!",1,2)[5]
```

Locate the point immediately preceding the third `chap` element in the document.

```
1:  start-point(//chap[3])
```

21 XLink

21.1 Goals and principles

The design of XLink [42] takes into account knowledge gained in established hypermedia systems and standards, in particular:

HTML [40] and [41] Defines several element types that represent links.

Hytime [17] Defines inline and out-of-line link structures and some semantic features, including traversal control and presentation of objects.

TEI [3] Provides structures for creating links, aggregate objects, and link collections out of them.

XLink provides a framework for creating both basic unidirectional links and more complex linking structures. It allows XML documents to:

- assert linking relationships among more than two resources;
- associate metadata with a link;
- create link databases that reside in a location separate from the linked resources.

21.2 A namespace and different kinds of links

XLink defines the following:

- A namespace characterizing xlink-type elements `http://wwww.w3.org/1999/xlink/`, e.g.,

```
1:  <myElement
2:    xmlns:xlink="http://www.w3.org/1999/xlink">
3:      ...
4:  </myElement>
```

- A set of elements and attributes anchored in that namespace. They define *linking elements*, characterized by a given linking semantic
  Three classes of such linking elements are defined.
  - *Simple link*. Outbound link between two participating resources (like HTML's `a` and `img`).
  - *Extended links*. Can have a quite complex structure, since they can be inbound or feature third-party arcs, or have an arbitrary numbers of participating resources. They can include elements for pointing to remote resources, elements for containing local resources, elements for specifying arc traversal rules, and elements for specifying human-readable resource and arc labels.

An XLink processor must be able to detect these contructs so that the application can handle them. XLink by itself does not does treat behaviour. This is up to the application layer using hints defined on the link elements.

21.3 XLink attributes

21.3.1 XLink core attributes

- `xlink:type` attribute indicates the type of the link:
  - value `simple` simple link
  - value `extended` extended link
- `xlink:href` this is the only locator attribute, and its value is an URI reference [15]
- `xlink:locator` are elements contained in extended links and defining resources participating in the link. Each locator holds an `xlink:role` attribute allowing to reference it.
- `xlink:arc` are elements contained in extended links and defining traversal behaviour using the following attributes:
  - `xlink:from` the source of the arc, the attribute value target a locator referenced by its `xlink:role` name
  - `xlink:to` the destination of the arc

21.3.2 XLink Behaviour properties

- `xlink:show` indicates expected behaviour on traversal:
  - `embed` link target should be included like <`img`>;

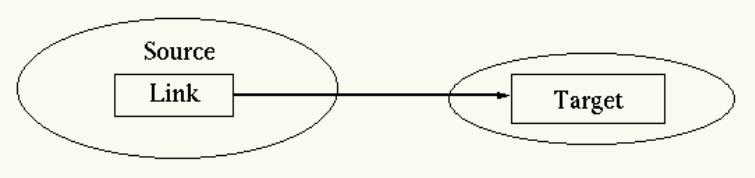

Figure 16: Graphic view of a simple link

  - `new` link target should create a new rendering, like opening a new window;
  - `replace` link target should replace resource containing link, like <A> in most cases.
- `xlink:actuate` indicates expected triggering of link:
  - value `onLoad` link activated when loading page, like <img>;
  - value `onRequest` link actiavted at user's request, like <a>.

21.3.3 XLink Semantic properties

- `xlink:title` provides human-readable text describing the links, simple links must use an attribute while extended links must use an element.
- `xlink:role` attribute containing a *qualified name* used to describe the function of the link's or locator's content.

21.4 Simple Links

Here are the limitations on simple links:

- the source of the link is the link element itself;
- there is only one target;
- the title is carried by a single attribute (unicity).

But they benefit from a lighter syntax compared to extended links.
A simple link pointing to a list of students could be used as follows:

```
1:  <studentlist xlink:href="students.xml">
2:    The list of students.
3:  </studentlist>
```

assuming the following element declaration:

```
1:  <!ELEMENT studentlist ANY>
2:  <!ATTLIST studentlist
3:   xmlns:xlink   CDATA    #FIXED "http://www.w3.org/1999/xlink/"
4:   xlink:type    (simple) #FIXED "simple"
5:   xlink:href    CDATA    #REQUIRED
6:   xlink:role    CDATA    #IMPLIED
7:   xlink:title   CDATA    #IMPLIED
8:   xlink:show    (new|embed|replace|undefined) "replace"
9:   xlink:actuate (onRequest|onLoad|undefined)  "onRequest"
10:  >
```

21.5 Extended Links

They have a set of *advanced* features:

- multiple sources and targets using `xlink:locator` elements;
- multiple actuation possible using `xlink:arc` elements;
- multiple `xlink:title` elements (I18N).

The following is an example of an extended link element

```
<!ELEMENT courseload (tooltip*, (student|course|audit|advisor|go*), gpa)>

<!ATTLIST courseload
          xmlns:xlink     CDATA           #FIXED "http://www.w3.org/1999/xlink"
          xlink:type      (extended)      #FIXED "extended"
          xlink:role      NMTOKEN         #FIXED "courseload"
          xlink:title     CDATA           #IMPLIED>

<!ATTLIST tooltip
          xlink:type      (title)         #FIXED "title"
          xlink:role      NMTOKEN         #FIXED "tooltip">
<!ATTLIST student
          xlink:type      (locator)       #FIXED "locator"
          xlink:role      NMTOKEN         #FIXED "student">
<!ATTLIST course
          xlink:type      (locator)       #FIXED "locator"
          xlink:role      NMTOKEN         #FIXED "course">
<!ATTLIST audit
          xlink:type      (locator)       #FIXED "locator"
          xlink:role      NMTOKEN         #FIXED "audit">
<!ATTLIST advisor
          xlink:type      (locator)       #FIXED "locator"
          xlink:role      NMTOKEN         #FIXED "advisor">
<!ATTLIST go
          xlink:type      (arc)           #FIXED "arc"
          xlink:role      NMTOKEN         #FIXED "go">
<!ATTLIST gpa
          xlink:type      (resource)      #FIXED "resource"
          xlink:role      NMTOKEN         #FIXED "gpa">
```

An XML document using these declarations might look something like this:

```
<courseload xlink:title="Course Load for Pat Jones">
 <go xlink:from="gpa" xlink:to="course" />
 <go xlink:from="student" xlink:to="course" />
 <go xlink:from="student" xlink:to="audit" />
 <go xlink:from="student" xlink:to="advisor" />
 <student xlink:href="..." />
 <course xlink:href="..." />
 <course xlink:href="..." />
 <audit xlink:href="..." />
 <advisor xlink:href="..." />
 <gpa>3.5</gpa>
</courseload>
```

The diagram in Figure 17 shows an extended link that associates five remote resources and provides rules for traversal among them. All of the arcs specified are third-party arcs; that is, the arcs go exclusively between remote resources. The nondirectional solid lines indicate that the link is associating the five resources; the dotted arrows indicate the traversal rules that the link provides.

The diagram shows directional traversal arcs reflecting the following settings, where both As and Cs are allowed to initiate traversal to all Bs:

```
<!-- go = arc-type -->
<go xlink:from="A" xlink:to="B" />
<go xlink:from="C" xlink:to="B" />
```

22 Scalable Vector Graphics (SVG)

22.1 Why another graphics format

22.1.1 Raster formats

The normal Web graphics that we use today.

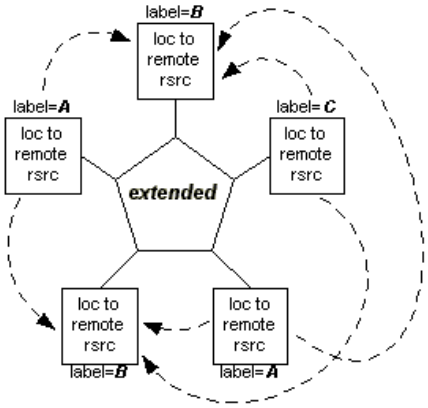

Figure 17: Out-of-link extended link with arcs

- JPEG: lossy, good for photos, yet no transparency;
- PNG: lossless, good for screenshots, allows transparency and has accurate color;
- GIF: widely used for animation.

With rasters graphics, *what you see is the only thing you will ever get.* Yet resolutions and sizes becoming more varied (mobile Web, Web on TV, high resolition LCD). And magnified bitmaps (zooms) are not really acceptable.

22.1.2 Vector formats

Scalable graphics let you zoom to any scale desired without loss of resolution.

- WebCGM: great for technical graphics, and has good HTML integration.

22.1.3 Authoring Web graphics today

- Create image in vector graphics drawing program;
- save as eps for print;
- rasterise at assumed screen resolution;
- add filter effects;
- guess at gamma (Mac, PC);
- grind down the colors;
- optimize for Web (iterative process).

22.2 Overview of SVG

Much existing graphical work starts as vectors. Yet rasterization for the Web means extra work. Often maintenance and localisation requires re-doing all the graphics, while different styles or sizes need near-duplicate images.

SVG's aim is to preserve investment, reduce costs, and speed download.

- Written in XML;
- specifies vector graphics for illustrations;
- resizable, for increasingly diverse displays;
- small, fast downloads;
- stylable with CSS, transform with XSLT;
- distributed symbol libraries ;

- compelling for web content authors;
- is in the final stages of completion at the W3C (last call).

22.2.1 Basic ingredients

SVG has all the normal vector graphics stuff. Yet its fundamental primitive is the *Graphical object*, not a point or line. Hence it is similar to most other textual markup.

- Polylines, Bézier curves, etc;
- plain, pattern, gradient fills;
- plain, marker, gradient strokes;
- templates/symbol libraries;
- transformational capabilities;
- inclusion of images;
- clipping;
- Raster effects.

22.2.2 Paths

- Simple or compound paths, closed or open;
- can be filled, stroked, marked, used for clipping;
- SVG *shapes* use paths for building common geometric shapes.

22.2.3 Transformations

- Default coordinate system is Y-down (origin at top left), one unit is one pixel;
- viewport maps an area in world coordinates to an area on screen;
- transformations alter the coordinate system (2x3 transformation matrix for computers, translate, rotate, scale, skew for humans).

22.2.4 SVG and XLink/XPointer

- *Re-use* of paths, symbols, markers, gradient fills...;
- `IDREF` only does local references;
- XPointer allows uniform local and Web-wide reference;
- site management tools; (need to identify links, we don't know all possible namespaces);
- XLink: A single link identification for XML.

22.2.5 Example

```
 1: <?xml version="1.0" standalone="yes"?>
 2: <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
 3:    "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
 4: <svg width="20cm" height="175mm">
 5:   <g style="stroke:black; fill:none; text-antialiasing:true;stroke-antialiasing:true" >
 6:     <text x="5" y="25" style="font-size:24">Exemple SVG :
 7:                                rectangles et ellipses</text>
 8:       <rect x="10"  y="50"  width="100" height="75"  />
 9:       <rect x="120" y="50"  width="100" height="75" style="fill:red" />
10:       <rect x="230" y="50"  width="100" height="75"
11:                    style="stroke:lime; stroke-width:6" />
12:       <rect x="340" y="50"  width="100" height="75"
13:             style="stroke:blue; fill:none; stroke-width:3;
14:             stroke-dasharray:10 5;stroke-linejoin:miter"  />
15:     </g>
16:     <g style="stroke:none; fill:blue">
17:       <rect x="10"  y="150"  width="100" height="75" rx="40" ry="40" />
18:       <rect x="120" y="150"  width="100" height="75" rx="50" ry="50" />
19:       <rect x="230" y="150"  width="100" height="75" rx="60" ry="60"
```

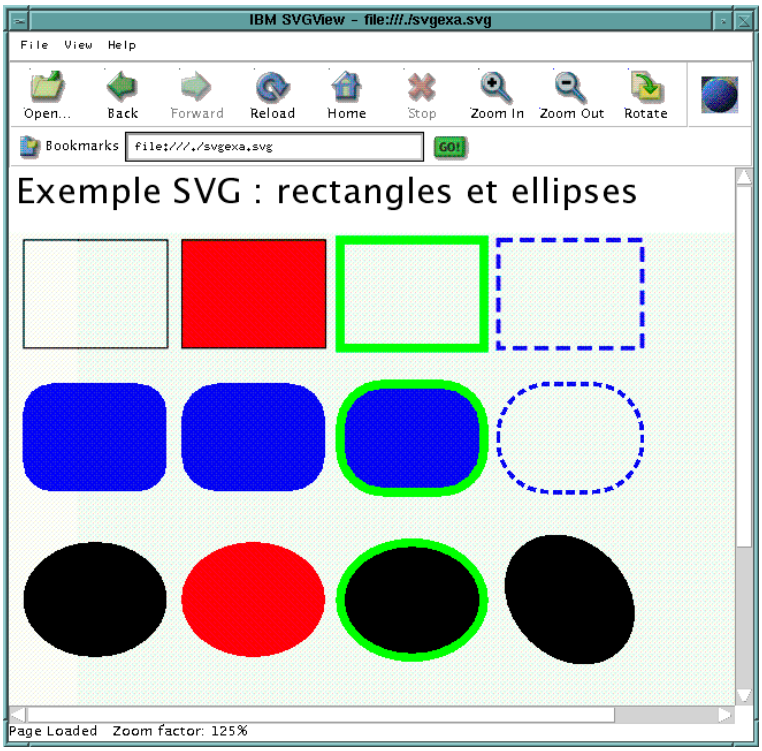

Figure 18: Simple graphics forms drawn with SVG

```
20:                     style="stroke:lime; stroke-width:6" />
21:        <rect x="340" y="150"  width="100" height="75" rx="70" ry="70"
22:              style="stroke:blue; fill:none; stroke-width:3;
23:              stroke-dasharray:6 3;stroke-linejoin:miter"  />
24:      </g>
25:      <ellipse cx="60"  cy="300" rx="50" ry="40"  />
26:      <ellipse cx="170" cy="300" rx="50" ry="40" style="fill:red" />
27:      <ellipse cx="280" cy="300" rx="50" ry="40"
28:               style="stroke:lime; stroke-width:6" />
29:      <ellipse cx="390" cy="300" rx="50" ry="40" angle="45"  />
30:  </svg>
```

Figure 18 is the result of interpreting the above SVG file by IBM's SVGView [14] SVG interpreter.

22.3 SVG and styling

22.3.1 Cascading Style Sheets (CSS)

- XML describes content and structure;
- CSS describes presentation;
- CSS makes pages that are faster to download, are more accessible, more maintainable, and more democratic since both users and authors can influence presentation.
- CSS is specifically designed for the Web, and it supports:
  - progressive rendering;
  - dynamic modification;
  - downloadable fonts;
  - well defined colors (sRGB ICC profile, but full gamut);
  - visual and aural rendering;
  - cascading reader/author balance.

An example of a CSS2 Media-specific style sheets follows.

```
 1:     BODY {
 2:       color: black;
 3:       background: white;
 4:     }
 5:
 6:  @media tv {
 7:     BODY {
 8:       color: white;
 9:       background: black;
10:     }
11:   }
```

22.3.2 Transformation styling

- Needed for on-demand generated graphics;
- can help produce accessible graphics for other media (speech, Braille);
- XSLT can transform source XML into SVG:
  - XSLT can switch namespace (to SVG);
  - need to use only *external* stylesheets;
  - tools exist to compile XSL-T into servlet.
- one can also use procedural (DOM) transformation in Perl, Java, Javascript, etc.

22.4 Advanced SVG features

22.4.1 Text in Graphics

Existing solutions for text in graphics are inadequate:

- text inside GIFs is a big problem;
- HTML `alt` & `longdesc` have lots of problems;
- Sliced images, CSS text, tables are a messy approach.

SVG allows genuine text to be used in graphics on the Web. SVG `title`, `desc` and `text` are:

- internationalized (any Unicode character is allowed);
- searchable, selectable and indexable;
- restylable;
- accessible for the print-impaired;
- easy to maintain;
- dynamically modifiable.

Text in SVG can be displayed on a path, can have gradient fills, be rotated, etc. Note, however, that there is *no text flow* in SVG (lines should be built by the application). Both horizontal (ltr, rtl) and vertical text are supported.

22.4.2 Fonts in SVG

CSS allows list of font families:

```
1:  font-family: hattenschweiller, tahoma, trebuchet,
2:               "gill sans", arial, helvetica, sans-serif;
```

WebFonts (TM) can be used for intelligent matching and font download.
SVG fonts are useful for various reasons:

- reliance on client installed fonts can compromise design;
- converting to outlines saves design;
- WebFont (TM) download works, but this format is not universally supported;

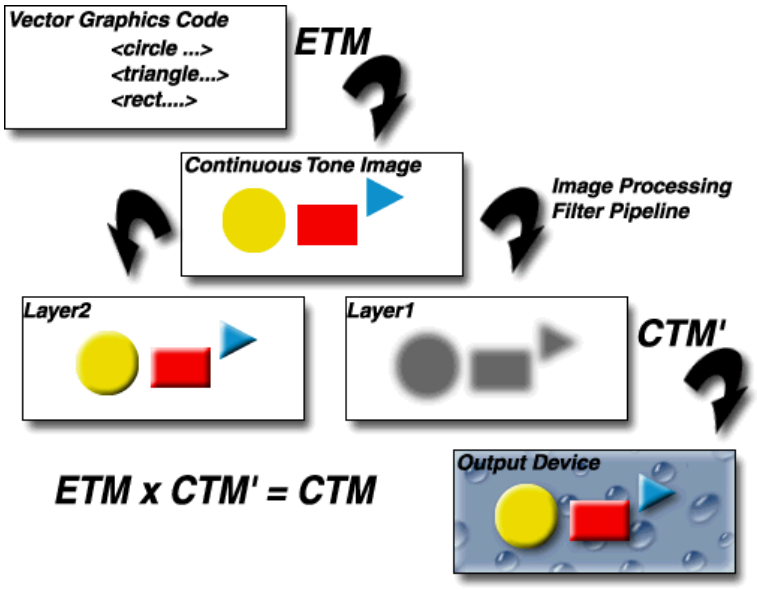

Figure 19: Flow of data through rendering pipeline

- SVG can already describe cubic and quadratic Béziers;
- SVG fonts describe glyphs, with kerning;
- SVG can include math expressions (MathML syntax);
- preserves exact glyph shape, preserves text as text.

22.4.3 SVG and Raster Images

- SVG can include JPEG and PNG images;
- size in pixels, user space or real-world units;
- resampling preserves scalability;
- images can be clipped, have transparent overlays.

SVG also allows several raster effects:

- client-side rasterisation of vector image;
- SVG adds intermediate, continuous tone rasterization phase;
- image processing nodes act on intermediate image;
- results of nodes can be composited and merged.

22.4.4 RDF for metadata in SVG

RDF can be used to describe the content of a graphic using metadata.

```
 1:  <?xml version="1.0" standalone="yes"?>
 2:  <svg width="10cm" height="7cm"
 3:      xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19990812.dtd'>
 4:      <desc>CERN Building Layout</desc>
 5:      <metadata>
 6:        <rdf:RDF
 7:                xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 8:                xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
 9:                xmlns:dc = "http://purl.org/dc/elements/1.0/"
10:                xmlns:svgmetadata = "http://www.w3.org/..." >
11:          <rdf:Description about=""
12:                   dc:title="CERN Building Layout"
```

```
13:                    dc:description="Map of the locations of all CERN buildings"
14:                    dc:publisher="CERN ST"
15:                    dc:date="2000-05-23"
16:                    dc:format="image/svg"
17:                    dc:language="en" >
18:             <dc:creator>
19:               <rdf:Bag>
20:                   <rdf:li>Pete</rdf:li>
21:                   <rdf:li>Renée</rdf:li>
22:               </rdf:Bag>
23:             </dc:creator>
24:             <svgmetadata:General MeetsAccessibilityGuidelines="true"/>
25:           </rdf:Description>
26:         </rdf:RDF>
27:       </metadata>
28:       <g><!-- graphic goes here --></g>
29:   </svg>
```

22.5 Animation and SVG

- Dynamic CSS binding: `:hover`, `cursor`, `:active`;
- DOM animation (Javascript, JAVA, etc) for dynamic manipulation, client-side generation;
- declarative animation of attributes and properties.

The W3C SYMM WG (which develops SMIL) and the SVG WG are collaborating on a core model for *timing* and for *animation* for XML. SVG is the first language to use this.

SMIL [37] has the following characteristics:

- Written in XML;
- specifies synchronisation of streaming and non-streaming media (images, audio, video, text);
- specifies layout (CSS subset) and *timing*;
- widely used in net players;
- new version (SMIL Boston) in development.

22.6 Summary for SVG

- A better way to do even raster images;
- maximises investment in XML, CSS, DOM,...;
- restylable and encourages graphics reuse;
- meets the challenge of Web device diversification;
- enhances Web accessibility;
- gives Web designers many new design capabilities.

23 Xschema

23.1 Highlights

XML Schemas are a tremendous advancement over DTDs:

- enhanced datatypes:
  - more than 40 versus 10;
  - can create your own datatypes;
  - can define the lexical representation, e.g., one can say that an element can contain strings of the form `ddd-dddd`, where `d` stands for a *digit*.
- written in XML, thus allowing the use of XML tools;
- DTDs cannot deal with namespaces, so that schema are essential in this respect;

- object-oriented design featuring inheritance for simple and complex types, that can be extended or restricted and new types can be derived from old ones;
- can express sets, where the child elements may occur in any order;
- can specify element content as being unique (keys on content) and uniqueness within a region;
- can define multiple elements with same name but different content;
- can define elements with null content;
- can create equivalent element classes, where one can be used in place of another;
- powerful wildcards (regular expressions in patters);
- all this allows for modular Schema construction;
- documentation mechanism.

A simple example follows. First the DTD syntax:

```
<!ELEMENT text (#PCDATA|emph|name)*>
<!ATTLIST text timestamp NMTOKEN #REQUIRED>
```

and then the same using XML Schema syntax:

```
<element name="text">
 <type content="mixed">
  <element ref="emph"/>
  <element ref="name"/>
  <attribute name="timestamp"
             type="date"
             minOccurs="1"/>
  </type>
</element>
```

23.2 The Schema Architecture: Static and dynamic

- A schema can be identified by a document, an application, or the user (on an equal footing);
- an XML Schema is well-formed XML and is valid with respect to the Schema DTD;
- the document is schema-valid w.r.t the schema;
- the schema is schema-valid w.r.t. the schema for schemas;
- a document is validated by a schema processor (XML application).

23.3 Status

- More than DTD functionality;
- contrain and document meaning, usage and relationships of the constituent parts of a document: datatypes, elements and their content, attributes and thier values;
- inheritance;
- data-friendly, with a good inventory of primitive datatypes;
- a tutorial [34] and two component documents (structures [35] and datatypes [36]) are in last call (May 2000);
- XML Schema is predicated on and layered on top of XML 1.0 well-formedness plus namespaces.

23.4 Complex types

23.4.1 Extension

Complex types can be *extended*. Here is the definition of the original type `name`:

```
<type name='name'>
 <element name='title'
          minOccurs='0'/>
 <element name='forename'
          minOccurs='0'
          maxOccurs='*'/>
 <element name='surname'/>
</type>
```

It is equivalent to the following DTD fragment

```
(title?,forename*,surname)
```

From this we can derive the type `fullName`, as follows:

```
<type name='fullName'
      source='name'
      derivedBy='extension'>
 <element name='genMark'
          minOccurs='0'/>
</type>
```

which is equivalent to the effective type:

```
<type name='fullName'>
 <element name='title'
          minOccurs='0'/>
 <element name='forename'
          minOccurs='0'
          maxOccurs='*'/>
 <element name='surname'/>
 <element name='genMark'
          minOccurs='0'/>
</type>
```

It is equivalent to the following DTD fragment:

```
(title?, forename*, surname, genMark?)
```

23.4.2 Restriction

Restriction for complex types is harder to handle syntactically, because of the significance of linear order in content models, but the semantics are completely parallel to the simple type case:

```
<type name='simpleName'
      source='name'
      derivedBy='restriction'>
 <restrictions>
  <element name='title'
           maxOccurs='0'/>
  <element name='forename'
           minOccurs='1'/>
 </restrictions>
</type>
```

Content model aspects not mentioned are left alone, including the "`maxOccurs='*'`" on <`forename`> and the whole particle for <`surname`>, so the effective content model of `simpleName` is:

```
<type name='simpleName'>
 <element name='title'
          maxOccurs='0'
          minOccurs='0'/>
<!-- i.e. forbidden -->
 <element name='forename'
          minOccurs='1'
          maxOccurs='*'/>
 <element name='surname'/>
</type>
```

It is equivalent to the following DTD fragment:

```
(forename+,surname)
```

23.4.3 Example of an instance

```
<name>
<title>Ms</title>
<surname>Steinem</surname>
</name>

  <name xsi:type='simpleName'>
   <foreName>Harry</foreName>
   <foreName>S</foreName>
   <surname>Truman</surname>
  </name>

  <name xsi:type='fullName'>
   <forename>Albert</forename>
   <surname>Einstein</surname>
   <genMark>Jr</genMark>
  </name>
```

23.4.4 Datatype example

The same type of derivation can also be used for data types. This has no equivalent in DTDs' Consider the simple type case first:

```
  <datatype name='bodytemp'
         source='decimal'>
   <precision value='4'/>
   <scale value='1'/>
   <minInclusive value='35.5'/>
   <maxInclusive value='39.5'/>
  </datatype>
```

A *derived type* from the above is:

```
  <datatype name='healthyBodytemp'
       source='bodytemp'>
   <maxInclusive value='36.7'/>
  </datatype>
```

The `healthyBodytemp` type definition is defined by closing down the permitted range of `bodytemp`. The type *inherits* the other *facets* of `bodytemp`, so that the complete type definition of healthyBodytemp is

```
  <datatype name='healthyBodytemp'
          source='decimal'>
   <precision value='4'/>
   <scale value='1'/>
   <minInclusive value='35.5'/>
   <maxInclusive value='36.7'/>
  </datatype>
```

23.5 Some of the built-in primitive datatypes

```
Data Types                    Examples
==========                    ========
string                        "Hello World"
boolean                       true, false, 1, 0
float, double                 -INF (infinity), -2.23E2, -123, -3, -0,
                              0, 4, 2.23E3, INF, NaN (not a number)
decimal                       3.14159
timeInstant, recurringInstant 1999-12-04T20:12:00.000 ('T' is date/time separator)
timeDuration                  1Y2M3DT10H30M12.3S
binary                        010010111001
uri                           http://www.mysite.org
language                      any valid "xml:lang" value, e.g., fr-CH, de
Name                          XML 1.0 name ("hello-there")
```

```
NCName                      XML namespace (un-qualified) name (part)
QNAME                       XML namespace (qualified) name (book:part)
ID                          XML 1.0 ID attribute type
IDREF                       XML 1.0 IDREF attribute type
IDREFS                      XML 1.0 IDREFS attribute type
ENTITY                      XML 1.0 ENTITY attribute type
ENTITIES                    XML 1.0 ENTITIES attribute type
NMTOKEN                     XML 1.0 NMTOKEN attribute type
NMTOKENS                    XML 1.0 NMTOKENS attribute type
NOTATION                    XML 1.0 NOTATION attribute type
integer                     456
non-negative-integer        zero to infinity
positive-integer            one to infinity
non-positive-integer        negative infinity to zero
negative-integer            negative infinity to negative one
date                        1999-05-21
time                        13:20:00.000
```

23.6 Regular expressions

Many of the primitive datatypes can have a regular expression as pattern facet. A few examples of regular expressions follow:

```
Regular Expression   Examples
==================   ========
Chapter \d           Chapter 1
a*b                  b, ab, aab, aaab,...
[xyz]b               xb, yb, zb
a?b                  b, ab
a+b                  ab, aab, aaab,...
[a-c]x               ax, bx, cx
[-ac]x               -x, ax, cx
[ac-]                ax, cx, -x
[^0-9]x              any non-digit char followed by x
\Dx                  any non-digit char followed by x
Chapter\s\d          Chapter followed by a blank followed by a digit
ho{2} there          hoho there
(ho\s){2} there      ho ho there
.abc                 any char followed by abc
(a|b)+x              ax, bx, aax, bbx, abx, bax,...
a{1,3}x              ax, aax, aaax
a{2,}x               aax, aaax, aaaax,...
\w\s\w               word (alphanumeric plus dash) followed by a space followed by a word
```

A pattern that chooses an IP address (four times a number between 0 and 255) is used in the following definition:

```
 1:  <datatype name="IP" source="string">
 2:      <pattern value="(([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
 3:                      ([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])"/>
 4:         <annotation>
 5:            <AppInfo>
 6:                Datatype for representing IP addresses.  Examples,
 7:                   129.83.64.255, 64.128.2.71, etc.
 8:                This datatype restricts each field of the IP address
 9:                to have a value between zero and 255, i.e.,
10:                   [0-255].[0-255].[0-255].[0-255]
11:            </appInfo>
12:         </annotation>
13:      </pattern>
14:  </datatype>
```

We constrain the contents of an element, e.g., an ISBN number, by defining a special type (`ISBNtype`):

```
1:      <datatype name="ISBNtype" source="string">
2:          <pattern value="\d{5}-\d{5}-\d{5}"/>
3:          <pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
4:          <pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
5:      </datatype>
```

`ISBNtype` has a string content conforming to one of the following three patterns:

1. 5 digits followed by a dash followed by 5 digits followed by another dash followed by 5 more digits.
2. 1 digit followed by a dash followed by 3 digits followed by another dash followed by 5 digits followed by another dash followed by 1 more digit.
3. 1 digit followed by a dash followed by 2 digits followed by another dash followed by 6 digits followed by another dash followed by 1 more digit.

23.7 Referencing a schema in an XML instance document

Consider the following document instance:

```
 1:  <?xml version="1.0"?>
 2:  <LaboratoryList
 3:    xmlns             ="http://www.heplabs.org/LaboratoryList"
 4:    xmlns:xsi         ="http://www.w3.org/1999/XMLSchema/instance"
 5:    xsi:schemaLocation="http://www.heplabs.org/LaboratoryList
 6:                        http://www.heplabs.org/LaboratoryList/LaboratoryList1.xsd">
 7:          <Laboratory>
 8:                  <Name>CERN</Name>
 9:                  <Location><town>Geneva</town><country>CH</country></Location>
10:                  <Created>1954</Created>
11:                  <Accelerators>
12:                    <MName>PS</MName><MEnergy>30</MEnergy><MDate>1959</MDate>
13:                    <MName>SPS</MName><MEnergy>450</MEnergy><MDate>1976</MDate>
14:                    <MName>LEP</MName><MEnergy>105</MEnergy><MDate>1989</MDate>
15:                    <MName>LHC</MName><MEnergy>105</MEnergy><MDate>2005</MDate>
16:                  </Accelerators>
17:          </Laboratory>
18:          ...
19:  </LaboratoryList>
```

The `LaboratoryList` (root) element contains various declarations. For instance, we declare that the `schemaLocation` attribute comes from the XML Schema Instance namespace (`xsi`). The value of schemaLocation is a pair of values: a namespace and the URI to a schema. When the XML parser processes this XML document it will use the schemaLocation pair of values to determine the XML Schema that it conforms to. It will retrieve the schema at the URI specified in schemaLocation (`LaboratoryList.xsd` here) and then it will open up this schema document to confirm that its targetNamespace value matches the namespace value shown in schemaLocation. In this case it does. Via the default namespace declaration we state that all the elements in the present XML instance document come from the same namespace.

24 MathML

24.1 Introduction

MathML will make the Web even better for educational, scientific and technical materials. It also has the potential to make mathematics accessible to those with visual disabilities. It will allow mathematical content to be reused and exchanged with technical computing systems for further manipulation. (W3C Director Tim Berners-Lee, 7 April, 1998)

24.2 Aims of MathML

- Encode mathematical material suitable for teaching and scientific communication at all levels;

- encode both mathematical notation and mathematical meaning;
- facilitate conversion to and from other math formats, both presentational and semantic. Output formats should include: graphical displays, speech synthesizers, computer algebra, other math layout languages (TEX), plain text displays (VT100 emulators), print media, braille. Such conversions may entail loss of information.
- allow information intended for specific renderers and applications;
- support efficient browsing for lengthy expressions;
- provide for extensibility;
- be well suited to template and other math editing techniques;
- be human legible, and simple for software to generate and process.

24.3 MathML: Present status

- Work on Mathml 2.0 is going on actively.
- Mozilla and Amaya render presentation MathML *natively* or you can use the techexplorer plugin or Webeq Applet to render in other browsers, but (currently) you then have to wrap the math tag in an Applet or object wrapper.
- Mathematica offers a reasonably complete MathML editing environment. You can also use MSword with `mathtype`, and other companies will follow soon...

24.4 MathML: presentation markup

- Describes mathematical notation structure;
- 28 elements, more than 50 attributes;
- generally, each presentation element corresponds to a single kind of "layout schema" (two-dimensional notational devide, such as row, superscript, underscript);
- variants are expressed via attributes.

24.4.1 List of the elements

- Token elements

  `<mi>` identifier
  `<mn>` number
  `<mo>` operator, fence, or separator
  `<mtext>` text
  `<mspace/>` space
  `<ms>` string literal

- *General layout schemata*

  `<mrow>` group any number of subexpressions horizontally
  `<mfrac>` form a fraction from two subexpressions
  `<msqrt>` form a square root sign (radical without an index)
  `<mroot>` form a radical with specified index
  `<mstyle>` style change
  `<merror>` enclose a syntax error message from a preprocessor
  `<mpadded>` adjust space around content
  `<mphantom>` make content invisible but preserve its size
  `<mfenced>` surround content with a pair of fences

- *Script and limit schemata*

  `<msub>` attach a subscript to a base
  `<msup>` attach a superscript to a base
  `<msubsup>` attach a subscript-superscript pair to a base
  `<munder>` attach an underscript to a base

<mover> attach an overscript to a base
<munderover> attach an underscript-overscript pair to a base
<mmultiscripts> attach prescripts and tensor indices to a base

- *Tables and matrices*

  <mtable> table or matrix
  <mtr> row in a table or matrix
  <mtd> one entry in a table or matrix
  <maligngroup/> alignment marker
  <malignmark/> alignment marker

- *Enlivening Expressions*

  <maction> bind actions to a subexpression

24.4.2 Examples of presentation attributes in the DTD

fontsize size of the font;
fontweight `normal`, or `bold`;
fontstyle `normal` or `italic`;
fontfamily name of the font;
color color expressed in some model;
form `prefix`, `infix`, `postfix`;
fence `true`, `false`;
separator `true`, `false`;
lspace left space;
rspace right space;
stretchy `true`, `false`;
symmetric `true`, `false`;
maxsize maximum size;
minsize minimum size;
largeop `true`, `false`;
movablelimits `true`, `false`;
accent `true`, `false`;
width width of structure;
height height of structure;
depth depth of structure.

24.5 MathML: content markup

Directly describe mathematical objects.

- about 75 elements and about a dozen attributes;
- *Token elements* <cn>, <ci>.
- *Basic content elements* <apply>, <reln>, <fn>, <interval>, <inverse>, <sep/>, <condition>, <declare>, <lambda>, <compose/>, <ident/>.
- *Arithmetic, algebra and logic* <quotient/>, <exp/>, <factorial/>, <divide/>, <max/>, <min/>, <minus/>, <plus/>, <power/>, <rem/>, <times/>, <root/>, <gcd/>, <and/>, <or/>, <xor/>, <not/>, <implies/>, <forall/>, <exists/>, <abs/>, <conjugate/>.
- *Relations* <eq/>, <neq/>, <gt/>, <lt/>, <geq/>, <leq/>.
- *Calculus* <ln>, <log>, <int/>, <diff/>, <partialdiff/>, <lowlimit>, <uplimit>, <bvar>, <degree>.
- *Theory of sets* <set>, <list>, <union/>, <intersect/>, <in/>, <notin/>, <subset/>, <prsubset/>, <notsubset/>, <notprsubset/>, <setdiff/>.
- *Sequences and series* <sum/>, <product/>, <limit/>, <tendsto/>.

$$x^2 + 4x + 4 = 0$$

$$A = \begin{bmatrix} x & y \\ z & w \end{bmatrix}$$

- *Trigonometry* <sin/>, <cos/>, <tan/>, <sec/>, <csc/>, <cot/>, <sinh/>, <cosh/>, <tanh/>, <sech/>, <csch/>, <coth/>, <arcsin/>, <arccos/>, <arctan/>.
- *Statistics* <mean>, <sdev>, <var>, <median>, <mode>, <moment>.
- *Linear algebra* <vector>, <matrix>, <matrixrow>, <determinant>, <transpose>, <select>.
- *Semantic mapping elements* <annotation>, <semantics>, <annotation-xml>.

24.6 MathML Examples

24.6.1 Presentation Markup

```
 1:  <mrow>
 2:    <mrow>
 3:      <msup>
 4:        <mi>x</mi>
 5:        <mn>2</mn>
 6:      </msup>
 7:      <mo>+</mo>
 8:      <mrow>
 9:        <mn>4</mn>
10:        <mo>&InvisibleTimes;</mo>
11:        <mi>x</mi>
12:      </mrow>
13:      <mo>+</mo>
14:      <mn>4</mn>
15:    </mrow>
16:    <mo>=</mo>
17:    <mn>0</mn>
18:  </mrow>
```

24.6.2 Content Markup

```
 1:  <reln>
 2:    <eq/>
 3:    <apply>
 4:      <plus/>
 5:      <apply>
 6:        <power/>
 7:        <ci>x</ci>
 8:        <cn>2</cn>
 9:      </apply>
10:      <apply>
11:        <times/>
12:        <cn>4</cn>
13:        <ci>x</ci>
14:      </apply>
15:      <cn>4</cn>
16:    </apply>
17:    <cn>0</cn>
18:  </reln>
```

24.6.3 Presentation Markup

```
1:  <mrow>
```

```
 2:    <mi>A</mi>
 3:    <mo>=</mo>
 4:    <mfenced open="[" close="]">
 5:      <mtable>
 6:        <mtr>
 7:          <mtd><mi>x</mi></mtd>
 8:          <mtd><mi>y</mi></mtd>
 9:        </mtr>
10:        <mtr>
11:          <mtd><mi>z</mi></mtd>
12:          <mtd><mi>w</mi></mtd>
13:        </mtr>
14:      </mtable>
15:    </mfenced>
16:  </mrow>
```

24.6.4 Content Markup

```
 1:  <reln>
 2:    <eq/>
 3:    <ci>A</ci>
 4:    <matrix>
 5:      <matrixrow>
 6:        <ci>x</ci>
 7:        <ci>y</ci>
 8:      </matrixrow>
 9:      <matrixrow>
10:        <ci>z</ci>
11:        <ci>w</ci>
12:      </matrixrow>
13:    </matrix>
14:  </reln>
```

24.7 Other bits and pieces about MathML

24.7.1 MathML interface

- MathML must work with wide variety of renderers, processors, translators and editors;
- embed MathML in HTML markup;
- make browser recognize MathML markup (semantics, namespace management, document validation);
- MathML must be rendered (native, plugin, Java applets);
- tools for authoring, editing, exporting MathML (computer algebra, TEX, WYSIWYG).
- intergration is with `<math>` tag.

```
1:       <META Content-math-Type="text/mathml">
2:           ....
3:       <math>
4:         <msup><mi>x</mi><mn>2</mn></msup>
5:       </math>
6:       <math type="text/mathml-rendererB">
7:         <mi>&alpha;<mi><mo>=</mo><mn>0.4</mn>
8:       </math>
```

24.7.2 Many characters...

Based on Unicode (many thousands of characters are already in the baseplane, others will go to one of the supplementary planes). They include as sources ISO 9573-13, TEX, supplementary characters from *Mathematica*, *Maple*.

- ISO Symbol Entity Sets

  `ISOAMSA` Added Math Symbols: Arrows
  `ISOAMSB` Added Math Symbols: Binary Operators

`ISOAMSC` Added Math Symbols: Delimiters
`ISOAMSN` Added Math Symbols: Negated Relations
`ISOAMSO` Added Math Symbols: Ordinary
`ISOAMSR` Added Math Symbols: Relations
`ISOTECH` General Technical
`ISOPUB` Publishing
`ISODIA` Diacritical Marks
`ISONUM` Numeric and Special Graphic
`ISOBOX` Box and Line Drawing
`MMALIAS` MathML Aliases
`MMEXTRA` MathML Additions

- ISO Math Font Entity Sets (variants for representing mathematics

`ISOGRK3` Greek Symbols
`ISOMSCR` Math Script Font
`ISOMOPF` Math Open Face (Blackboard) Font
`ISOMFRK` Math Fraktur (Gothic) Font

- Other ISO Font Entity Sets

`ISOGRK1` Greek Letters
`ISOGRK2` Monotoniko Greek
`ISOGRK4` Alternative (emboldened) Greek Symbols
`ISOCYR1` Russian Cyrillic
`ISOCYR2` Non-Russian Cyrillic

25 XHTML

25.1 HTML's childhood

- The genesis of HTML was firmly rooted in simplicity;
- NCSA's Mosaic popularized the Web;
- most people don't think structurally;
- tags as formatting instructions;
- Tag Soup;
- standardization efforts attempt to stem the tide.

25.2 The Challenges facing HTML

- Prevalence of sloppy markup practices;
- patchy implementations of style sheet support;
- new kinds of browsers: Digital TVs, handhelds, phones and cars;
- pressure to subset HTML for simple clients;
- pressure to extend HTML for richer clients;
- call for improvements to HTML Forms;
- combining HTML with other tag sets: Vector Graphics, Animation, Multimedia, E-commerce, Math, Metadata, ...

25.3 W3C HTML Activity

- Earlier work in IETF on HTML 2.0, tables, internationalization and file upload;
- failure to make progress on HTML 3.0 (aka *HTML+*);
- Fall '95 W3C brings vendors together, HTML working group set up;
- Jan '97 W3C releases HTML 3.2 Recommendation;
- Dec '97 W3C releases HTML 4.0 Recommendation, HTML working group closes;

- May ’98 W3C workshop on Future of HTML leads to new HTML working group.

25.4 Scope of HTML Activity

The HTML activity is described in detail at the W3C site [30], so we mention only the main points.

- Maintenance of HTML 4.0, limited to fixing bugs and folding in errata;
- guidelines and tools for transforming HTML into XML;
- transition strategies - allowing existing browsers to successfully cope with HTML as XML;
- reformulation of HTML as a suite of modular components suitable for subsetting or combining with other tag sets;
- an improved match to database/workflow applications, mobile & embedded devices, television and ultra-thin clients for e-commerce, etc.;
- the development of a mechanism for specifying document profiles as a basis for authoring documents for given classes of browsers.

25.5 Moving to XML

- HTML was created as an application of SGML - the Standard Generalized Markup Language (ISO 8879:1986);
- XML is a descendant of SGML, which is easier to implement;
- XML requires you to:
  - make tags case-sensitive;
  - include end tags e.g. `</p>` and `</li>`;
  - add a / to empty tags, e.g. `<br />` and `<hr />`;
  - quote all attribute values, e.g. `<img src="karen.jpg" />`;

  ;
- these make it practical to parse well-formed XML without *a priori* knowledge of the tags;
- XHTML uses *lower-case* for tags and attributes;
- old browsers can render XHTML 1.0 provided you follow simple guidelines.

25.6 XHTML 1.0

- Defines a single namespace for html: `http://www.w3.org/1999/xhtml`;
- relies on HTML 4.01 for the semantics and data types of elements and attributes;
- defines XHML 1.0 DTDs corresponding to HTML4’s strict, transitional and framset DTDs;
- provides guidelines for authoring XHTML documents for delivery to existing Web browsers;
- existing HTML can be trivially converted to XHTML using W3C’s Tidy utility.

25.7 HTML Tidy

- W3C Open Source utility for fixing up HTML;
- Tidy corrects the markup in a way that matches where possible the observed rendering in popular browsers;
- can convert presentation markup to CSS rules;
- knows about ASP and PHP and can be taught about new tags, e.g. *Cold Fusion*;
- does a good job on cleaning up the HTML exported from Microsoft Word’97 and Word’2000;
- makes it trivial to roll content over to XHTML;
- Tidy is available on most platforms and has been integrated into numerous authoring environments.

25.8 Modularising HTML

- One-size-fits-all no longer holds. A cellphone can’t offer the same experience as a desktop machine;
- XHTML is designed for subsetting and extending;

- XHTML Modules form indivisible units for this purpose;
- initial set of modules derived from XHTML 1.0 (HTML4);
- new stuff includes Ruby, DOM2 Events, XForms;
- major application is for Television and Mobile domains;
- modules make it easier to define new document types that mix XHTML with other tag sets for vector graphics, animation, multimedia, math and more.;
- XHTML modules are being formalized using XML 1.0 DTDs and XML Schemas.

25.9 Basic Modules

All XHTML document types must include these modules:

- Structure - `body, div, head, html, span, title`;
- Basic Text - `h1..h6, p, pre, br, em, strong, ...`;
- Hypertext - `<a href="..."> ... </a>`;
- Lists - `dl, ol, ul, li`.

25.10 Additional Modules

- Presentation - `b, i, tt, big, small, sub, sup, hr`;
- Edits - `del, ins`;
- Ruby - *phonetic annotations for ideographic characters*;
- Forms *split across two modules*;
- Tables *split across two modules*;
- Images *with separate image map modules*;
- StyleSheets, Scripting, Intrinsic events;
- Applets, Object, Frames, Metadata etc.;
- Events - *new module for DOM2 events*.

25.11 XHTML Event Module

- New event module with markup for declaring events and handlers;
- designed to support DOM level 2 event bubbling model;
- events handlers are tied to elements in markup tree;
- events can be intercepted by ancestor elements either before or after being handled by child elements;
- user interface events, mutation events, and trigger events generated from scripts.

25.12 XForms - the next generation of Web forms

- HTML Forms introduced in 1993;
- critical part of the Web and closely coupled to scripting for application user interface;
- existing design is outdated;
- HTML charter calls for an improved match to workflow and database applications;
- forms are relevant to business applications, consumer applications and device control panels;
- requirements draft published in July 1999;
- work is underway to meet these requirements.

25.13 Key Goals for XForms

- Support for handheld, television, and desktop browsers, plus printers and scanners;
- richer user interface to meet the needs of business, consumer and device control applications;
- decoupled data, logic and presentation;
- improved internationalization;
- support for structured form data;

- advanced forms logic;
- multiple forms per page, and pages per form;
- suspend and Resume support;
- seamless integration with other XML tag sets.

25.14 XForms - Data, Logic and Presentation

25.14.1 Data

Builds on XML Schema to provide a small number of intrinsic data types plus the means to declare new types. These are used to declare the form's data model. XForms is targetted at a less skilled audience than XML Schemas.

25.14.2 Logic

Leverages familarity with spreadsheets and existing electronic forms packages: required fields, dependent fields, running totals, etc.

25.14.3 Presentation

A combination of style properties and presentation elements that bind to the form's data.

- Handheld computer vis a vis a printed form - different presentations;
- precise rendering is often important, and may be a legal requirement;
- *painted* layout where every widget is independently positioned;
- autolayout where UI is generated as needed.

This allows multiple views onto the same data.

25.15 Document Profiles

- As vendors introduce new versions of browsers, content developers have had to get to grips with a variety of differences between browsers from the same vendor as well as the larger differences between vendors.
- The range of browser platforms is rapidly expanding to include television sets, handheld organizers, cell phones, in-car systems and regular phones. Each of these platforms presents different capabilities.
- Authors want guarantees for how their content is rendered in an increasingly diverse environment;

W3C's solution involves:

- Characterising browser capabilities;
- Document profiles for authoring content;
- Selecting/Transforming content to match;

25.16 XHTML Modules as XML 1.0 DTDs

XHTML modules have been formalized as DTDs, where heavy use is made of parameter entities, using the following suffices:

`.mod` entities identifying DTD modules (files);
`.module` entities used to control the inclusion of DTD modules;
`.content` entities defining element content models;
`.class` used for elements from the same class;
`.mix` used for elements from different classes;
`.attrib` used for one or more attribute specifications.

25.17 XHTML elements

These are classified into the following sets:

`structural` elements used for the document's overall structure;
`block` elements that force line breaks;
`inline` elements rendered inline within enclosing blocks;
`phrasal` inline elements such as *em*> and *strong*;
`presentational` elements such as *i* and *b* etc.;
`special case ("feature")` used for linking, forms, etc.

25.18 Defining a new module for XHTML

A module can (same as with DocBook and TEI):

- add new attributes to existing elements by adding a new `ATTLIST` declaration;
- add new elements into existing classes by overriding a `.class` or `.mix` entity;
- override the content model for an existing element by overriding a `.content` entity;
- add new character entities.

25.19 Adding attributes to an existing element

You just need to include an `ATTLIST` for the attribute:

```
<!ATTLIST p
    myattr  CDATA   #IMPLIED
>
```

This works because XML allows multiple `ATTLIST` declarations for each element.

25.20 Adding new elements

You need to include an `ELEMENT` and an `ATTLIST`:

```
<!ELEMENT myelement ( #CDATA | myotherelement )* >

<!ATTLIST myelement
    myattr  CDATA   #IMPLIED
>

<!ELEMENT myotherelement EMPTY >
```

You also need to specify how the new element fits into XHTML by overriding a `.content`, `.class` or `.mix` entity, e.g.

```
<!ENTITY % Misc.class
    "ins | del | script | noscript | myelement">
```

25.21 Creating a new DTD

Having defined any new modules, you can now define a DTD for your new document type.

```
<!ENTITY % MyModule
    PUBLIC "-//CERN/Extensions 1.0//EN"
    "http://www.cern.ch/DTD/extensions1_0.dtd">

%MyModule;

<!ENTITY % XHTML11-strict
    PUBLIC "-//W3C/XHTML 1.1 Strict"//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11-s.dtd">

%XHTML11-strict;
```

25.22 Creating a subset of XHTML

You can drop XHTML modules by setting their `.module` entities to *ignore*, for instance, the following drops the more advanced features for tables:

```
<!ENTITY % XHTML1-table.module" "IGNORE">

<!ENTITY % XHTML11-strict
    PUBLIC "-//W3C/XHTML 1.1 Strict"//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11-s.dtd">

%XHTML11-strict;
```

25.23 Using the new DTD

Once declared, you can use it by referencing the DTD in the `DOCTYPE`:

```
<!DOCTYPE html PUBLIC "-//CERN/Extensions 1.0//EN"
    "http://www.cern.ch/DTD/extensions1_0.dtd">
<html>
  <head>
    <title>XHTML with Widgets!</title>
  </head>
  <body>
    <p>This is an example document using the
    nice extensions of cern.ch:</p>

    <widget type="spreadsheet"> ... </widget>
  </body>
</html>
```

PartIV
XML at CERN and in HEP

- An XML based Import/Export System for CERN's EDMS (Bertrand Rousseau/IT-EDMS);
- XML and Oracle (Maciej Marczukajtis/IT-DB);
- XML Detector Description for HEP (Radovan Chytracek/LHCb);
- RDF: a structured metadata model for Outreach (Roberta Faggian/ETT-DH);
- XML as a central strategy for handling scientific documents (Michel Goossens/CERN-IT, EU TIPS Project).

26 Document strategies for the Web

26.1 Storing huge knowledge bases

- Millions of document exist in electronic form. *Reuse* of the large investment in this knowledge base is an important point.
- *New* documents can exploit the advantages of the latest technologies and adapt readily to the target audience (e-commerce, scientific, dictionaries, etc.).
- *Several presentations* of the same source are often a must, and *ad hoc* and different techniques have to be used to optimally exploit the display possibilities of the medium (browser, print, audio-video).
- There exist a *variety* of application domains. Scientific articles, financial data, or ancient Greek poetry need different tools and target different audience.

26.2 XML as central source repository

- The developers of XML have taken into account the lessons learned in the last decade using SGML and HTML.
- By construction XML is (should be) an ideal tool for dealing with most kinds of data and (multilingual) source documents (based on Unicode, seamless integration with modern languages, such as Java, perl, and python).
- XML is supported by all major players in the Internet world: *Open Source* people as well as commercial vendors.
- Many free (and not-so-free) tools are available for all conceivable operating systems and purposes. Soon HTML-only browsers will only be a (bad) memory, and most Internet tools will support XML natively.

26.3 Handling scientific documents

Figure 20 shows the central position XML plays as a document stategy for the Web. At the top right we show the XML document, with its DTD (XML Schema) that can be transformed into TEX for typesetting via the procedures outlined at the top (`xmltex`, X2L, and X2F, then PassiveTEX with `xmltex`, three ways to go from XML to TEX [5]). We can also translate the document into HTML for viewing with present-day browsers (X2H via XSL). In both cases (TEX and HTML generation) also DSSSL and Jade can be used, although that method is more complex. In the (near) future, once browsers are able to handle XML directly, we can probably skip the HTML intermediate format and let CSS/XSLT style the XML file directly for display on the Web. We have also shown arrows going from left (TEX) to right (XML/HTML, browsers). Indeed, via programs (L2H) we want to transform the thousands of existing LATEX source documents into XML (using one or more standard DTDs) to store the information for archiving purposes. The vertical ellipse in the centre represents other editing tools, such as Adobe's *FrameMaker*, Microsoft's *Word*, and Corel's *WordPerfect*. They are expected to allow import/export of XML documents. Thus, XML will soon become the central element in a global strategy for managing electronic documents by allowing information to be stored, saved, shared, and used by different applications on all computer platforms.

26.4 Tools for innovative publishing in Science

To find the best way to produce and dissiminate scientific information on the Web an international collaboration [23] was set up, with HEP as the driving force (SISSA, CERN) and sponsored in part by the European Union. We plan to develop a set of user-friendly and advanced tools and services that are organized in an open system to support research information production, management, access, and use in a coherent manner. To study the model the proposed tools and services will first be integrated on a Web-based portal targeted at the high-energy physics community. The proposed system will support the activities of document writing, reviewing, publishing, searching, disseminating and reading, as well as the communication among members of the research community. The system should support a more productive research community, in which researchers can work in a more effective, inexpensive, and pleasant way: delays and costs due to paper documents are considerably reduced, multimedia can be added to electronic documents, information access can be improved (and information overload decreased) by using advanced information retrieval and filtering techniques.

26.5 PassiveTEX: TEX brings typography to XML

- Often typographic quality is a *must*.
- The *print* button of most present-day HTML browsers does not in general give high quality printable copy.
- *Special* and separate procedures are used to prepare the printable output for XML documents.
- PassiveTEX is a library of TEX macros which can be used to process an XML document which results from an XSL transformation to formatting objects.

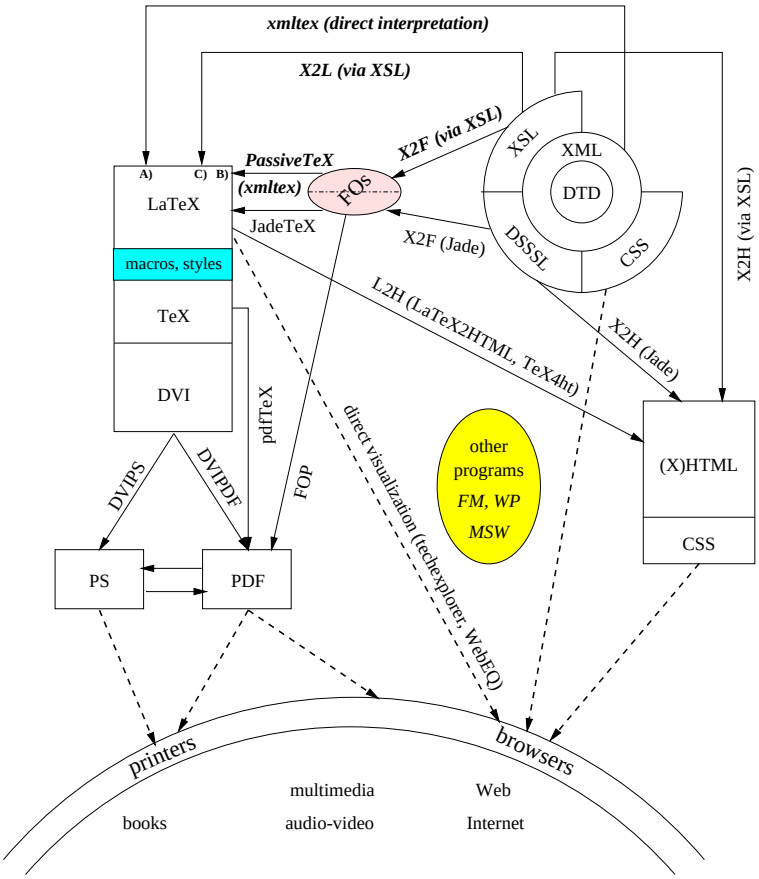

Figure 20: XML as the central part of a document strategy for the Web

- Running PassiveTEX with the pdfTEX variant of TEX generates high-quality PDF files in a single operation.
- PassiveTEX shows how TEX can remain the formatter of choice for XML, while hiding the details of its operation from the user.
- PassiveTEX derives from and builds on:
  - The `xmltex` [4] TEX package by David Carlisle, providing the core XML parser and UTF8 handler;
  - The JadeTEX package by Sebastian Rahtz which implemented the output of the Jade DSSSL processor's TEX backend, in conjunction with the catalogue of Unicode/TEX mappings built up for JadeTEX.
- Some work has gone on to support some standard DTDs:
  - *Text Encoding Initiative (TEI).*
    Using `teixlite.dtd` and the corresponding XSL stylesheets (See [3] and [21]) one can transform and typeset TEI marked up documents into FO (and HTML)
  - *Docbook.*
    Using the DocBook DTD and the corresponsing HTML and FO stylesheets (see [25]) one can transform and typeset DocBook markup up documents.

26.6 Advantages of PassiveTEX

- Rapid development since TEX takes care of the typesetting;
- TEX is a well-understood, robust, free page formatter;

- fonts, graphics inclusion, hyperlinks, etc,, come ‘for free’;
- mature handling of language issues, including hyphenation;
- high-quality math rendering;
- pdfTEX variant generates very high-quality PDF.

26.7 Disadvantages of PassiveTEX

- We are constrained to use TEX’s page makeup model, and have to force XSL FO to fit it;
- because LATEX is already high-level markup, it is too easy to allow things to fall through and take LATEX defaults;
- TEX macro writing is obscure and difficult and thus the system is not transparent for most programmers;
- TEX is large and monolithic, and unsuited to embedding in other applications.

26.8 `xmltex`

- Non validating (and not 100% conforming) namespace-aware XML parser implemented in TEX;
- reads encoding declaration and tries to implement it;
- reads DTD subset and expands entities properly;
- uses namespace to load modules of TEX code to process elements;
- handles UTF8 encoding;
- limited access to child and parent elements.

An example of `xmltex` instantiation code follows.

```
\XMLelement{emph}{}
 {\itshape}
 {}
%%% A table
\XMLelement{table}
 {\XMLattribute{rend}{\tablerend}{}}
 {\begin{longtable}{*{99}{l}}}
 {\end{longtable}}

\XMLelement{row}
 {}
 {\xmlgrab}
 {\gdef\temp{#1\\}\aftergroup\temp}
%%% And some MathML
\DeclareNamespace{m}{http://www.w3.org/1998/Math/MathML}

\XMLname{formula}{\FORMULA}

\XMLelement{m:math}
  {}
  {\ifthenelse{\equal{\XML@parent}{\FORMULA}}{}{\[}}
  {\ifthenelse{\equal{\XML@parent}{\FORMULA}}{}{\]}}

\XMLelement{m:mn}
  {}
  {\xmlgrab}
  {\mathrm{#1}}

\XMLelement{m:msqrt}
  {}
  {\xmlgrab}
  {\sqrt{#1}}
```

26.9 Present PassiveTEX support for XSL FO

26.9.1 Parts of XSL FO which are implemented fairly well

- blocks;
- inline sequences;
- lists;
- graphics inclusion;
- floats;
- font properties;
- links.

`xmltex` supports processing instructions to manipulate TEX formatting directly.

26.9.2 Parts of XSL FO implemented dubiously

- page specifications;
- tables and table properties;
- margins, borders, padding;
- pagination and page properties.

26.9.3 Parts of XSL FO which are not implemented at all

- bidi handling;
- backgrounds;
- aural properties;
- vast numbers of other assorted properties, especially shorthands, like `font="Times Roman 11pt bold"`.

26.10 Math rendering

PassiveTEX supports MathML directly. An XSL style sheet can pass `<math>` and its children through unchanged:

```
 1:  <xsl:template match="math">
 2:    <xsl:apply-templates mode="math"/>
 3:  </xsl:template>
 4:  <xsl:template    mode="math">
 5:    match="*|@*|comment()|processing-instruction()|text()"
 6:    <xsl:copy>
 7:    <xsl:apply-templates    mode="math"
 8:     select="*|@*|processing-instruction()|text()"/>
 9:   </xsl:copy>
10:  </xsl:template>
```

26.11 Final remarks on PassiveTEX

- No use is made of LATEX high-level constructs. No sections, no lists, no cross-refs, no bibliographies;
- XSL FO's underlying character set is Unicode; by default, entities are mapped to their Unicode position;
- all vertical and horizontal space is explicit in the specification;
- page and line breaking is left to TEX: the rest is up to the user;
- MathML handling not complete yet;
- more property values need to be supported (e.g., colors, fonts);
- more work needed on complex tables;
- SVG needs to be supported somehow (direct interpretation, pre-processing, MetaPost?), and SVG fragments need to be recognized directly to perform in-line graphical functions (e.g., setting text at an angle);

- try to use of the Unicode-based TEX variant (Omega) to handle non-Latin material more naturally.

26.12 Example of a scientific document

- original source is LATEX (see Figure 21;
- translated to XML with TEX4ht using LATEX-like *ad hoc* DTD and MathML;
- an XSL style sheet treats the *textual elements* of the document;
- math components are *passed through* unchanged to the back-end;
- MathML is interpreted directly in PassiveTEX.

The first part of the XML, mainly containing the MathML source corresponding to the LATEX typeset document shown in Figure 21 is shown below. For those with some knowledge of MathML it is not difficult to recognize the code for the formulae shown.

```
<section id="vavref">
<stitle>Vavilov theory</stitle>

<par>Vavilov<cite refid="bib-VAVI"/> derived a more accurate
straggling distribution by introducing the kinematic limit on the
maximum transferable energy in a single collision, rather than using
<inlinemath><math><msub><mi>E</mi><mrow><mtext>max</mtext></mrow></msub>
<mo>=</mo><mi>&infin;</mi></math></inlinemath>.

Now we can write<cite refid="bib-SCH1"/>:

<eqnarray ><subeqn><math><mi>f</mi> <mfenced open='(' close=')'>
<mi>&epsi;</mi><mo>,</mo><mi>&delta;</mi><mi>s</mi></mfenced>
<mo>=</mo> <mfrac><mrow><mn>1</mn></mrow>
<mrow><mi>&xi;</mi></mrow>
</mfrac><msub><mi>&phi;</mi><mrow><mi>v</mi></mrow></msub>
<mfenced open='(' close=')'>
<msub><mi>&lambda;</mi><mrow><mi>v</mi></mrow></msub><mo>,</mo>
<mi>&kappa;</mi><mo>,</mo><msup><mi>&beta;</mi><mrow><mn>2</mn></mrow>
</msup></mfenced></math></subeqn></eqnarray>
where
<eqnarray><subeqn><math><msub><mi>&phi;</mi><mrow><mi>v</mi></mrow></msub>
<mfenced open='(' close=')'>
<msub><mi>&lambda;</mi><mrow><mi>v</mi></mrow></msub><mo>,</mo>
<mi>&kappa;</mi><mo>,</mo>
<msup><mi>&beta;</mi><mrow><mn>2</mn></mrow></msup></mfenced>
  <mo>=</mo>
<mfrac><mrow><mn>1</mn></mrow>
       <mrow><mn>2</mn><mi>&pi;</mi><mi>i</mi></mrow>
</mfrac>
<msubsup><mo>&int;</mo>
<mrow><mi>c</mi><mo>+</mo><mi>i</mi><mi>&infin;</mi></mrow>
<mrow><mi>c</mi><mo>-</mo><mi>i</mi><mi>&infin;</mi></mrow></msubsup>
<mi>&phi;</mi><mfenced open='(' close=')'><mi>s</mi></mfenced>
<msup><mi>e</mi><mrow><mi>&lambda;</mi><mi>s</mi></mrow></msup>
<mi>d</mi><mi>s</mi><mspace width='2cm'/><mi>c</mi><mo>&geq;</mo><mn>0</mn>
                </math></subeqn>

<subeqn><math><mi>&phi;</mi><mfenced open='(' close=')'><mi>s</mi></mfenced>
<mo>=</mo><mo>exp</mo><mfenced open='[' close=']'><mi>&kappa;</mi>
<mrow><mo>(</mo><mn>1</mn><mo>+</mo><msup><mi>&beta;</mi>
      <mrow><mn>2</mn></mrow></msup><mi>&gamma;</mi><mo>)</mo></mrow>
</mfenced><mo>exp</mo><mfenced open='[' close=']'><mi>&psi;</mi>
<mfenced open='(' close=')'><mi>s</mi></mfenced></mfenced>
<mo>,</mo> </math></subeqn>
```

3 Vavilov theory

Vavilov[5] derived a more accurate straggling distribution by introducing the kinematic limit on the maximum transferable energy in a single collision, rather than using $E_{\max} = \infty$. Now we can write[2]:

$$f(\epsilon, \delta s) = \frac{1}{\xi}\phi_v\left(\lambda_v, \kappa, \beta^2\right)$$

where

$$\begin{aligned}
\phi_v\left(\lambda_v, \kappa, \beta^2\right) &= \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty}\phi(s)\, e^{\lambda s} ds \qquad\qquad c \geq 0 \\
\phi(s) &= \exp\left[\kappa(1+\beta^2\gamma)\right]\ \exp\left[\psi(s)\right], \\
\psi(s) &= s\ln\kappa + (s+\beta^2\kappa)\left[\ln(s/\kappa) + E_1(s/\kappa)\right] - \kappa e^{-s/\kappa},
\end{aligned}$$

and

$$\begin{aligned}
E_1(z) &= \int_z^\infty t^{-1}e^{-t}dt \qquad \text{(the exponential integral)} \\
\lambda_v &= \kappa\left[\frac{\epsilon - \bar{\epsilon}}{\xi} - \gamma' - \beta^2\right]
\end{aligned}$$

The Vavilov parameters are simply related to the Landau parameter by $\lambda_L = \lambda_v/\kappa - \ln\kappa$. It can be shown that as $\kappa \to 0$, the distribution of the variable λ_L approaches that of Landau. For $\kappa \leq 0.01$ the two distributions are already practically identical. Contrary to what many textbooks report, the Vavilov distribution *does not* approximate the Landau distribution for small κ, but rather the distribution of λ_L defined above tends to the distribution of the true λ from the Landau density function. Thus the routine `GVAVIV` samples the variable λ_L rather than λ_v. For $\kappa \geq 10$ the Vavilov distribution tends to a Gaussian distribution (see next section).

Figure 21: The document formatted by LaTeX

PartV
General conclusion

27 With XML towards a multi-cultural, multi-lingual, inter-disciplinary Web

27.1 XML is ASCII for the 21st century

- ASCII (ISO 646) solved a fundamental interchange problem for flat (English) text documents;
- Unicode/ISO 10646 extends that solution to the whole world;
- XML does the same for simple tree-structured documents;
- XML is a markup language used for *transferring (structured) data* that are described by a *data model*;
- XML has been a runaway success, on a much greater scale than its designers anticipated, not because separation of form from content is *right* , but because *data must travel the Web*;
- tree-structured documents are a useable transfer syntax for just about anything.

XML has defined a transfer syntax for tree-structured documents. Many data-oriented applications are being defined which build their own data structures on top of an XML document layer, effectively using XML documents as a transfer mechanism for structured data.

27.2 Final Summary

- XML allows you to construct your own markup language, optimized for the task at hand. You are free to use your own element, entity and attribute names in your native language and use and encoding

adapted to your local computer system;
- lots of tools are freely (and not so freely) available (Open source spirit);
- most major Internet players have adopted the standard;
- "Standard" DTDs are already being prepared by various scientific, commercial, etc., communities, and this will improve communication and inter-operability.
- XSL is a good companion tool for styling and transforming XML sources and it is well integrated with the Web (formatting objects, SVG,...);
- preparing scientific documents for the Web is a non trivial task, especially if they contain mathematical information.
- W3C sponsored specifications in many areas (SVG, XML Schema, XLink, XPointer, MathML, etc.) will ensure that all activities on the Web can be seamlessly integrated, so that investing in XML technology is a winning strategy in each case if the Web is your main communication channel.

References

[1] Apache XML Project. *FOP, XSL Formatting Object Processor in Java.* `http://xml.apache.org/fop/`

[2] Apache XML Project. *Xerces Java parser (C++ and perl implementations also exist).* `http://xml.apache.org/xerces-j/index.html`

[3] Lou Burnard and C.M. Sperberg-McQueen. TEI Guidelines for Electronic Text Encoding and Interchange. `http://etext.lib.virginia.edu/TEI.html`

[4] David Carlisle. *xmltex A non validating (and not 100% conforming) namespace aware XML parser implemented in TEX*. Available on CTAN in the directory `macros/xmltex/`.

[5] David Carlisle, Michel Goossens and Sebastian Rahtz. De XML à PDF via `xmltex`, XSLT et PassiveTEX. *Cahiers GUTenberg*, 35-36, pages 79-114, May 2000. (See also `http://www.gutenberg.eu.org/pub/GUTenberg/publications/cahiers.html#Cahier35-36`).

[6] James Clark. *xt, an implementation in Java of XSL Transformations.* `http://www.jclark.com/xml/xt.html`

[7] James Clark. *DSSSL implmentation.* `http://www.jclark.com/jade`

[8] European Union Web site. *Countries and currencies (A-K).* `http://europa.eu.int/comm/translation/currencies/entable1.htm`

[9] Michel Goossens and Sebastian Rahtz. *The LATEX Graphics Companion*. Addison-Wesley, Reading, 1997.

[10] Michel Goossens and Sebastian Rahtz. *The LATEX Web Companion*. Addison-Wesley, Reading, 1999.

[11] Tony Graham. *Unicode: A Primer*. M & T Books (IDG), Foster City, 2000.

[12] Yannis Haralambous and John Plaice. The latest developments in Omega. *TUGBoat*, 17 (2), pages 181-183, June 1996. (See also `http://www.gutenberg.eu.org/omega/`).

[13] Elliotte Rusty Harold. *Cafe con Leche XML News, and Resources.* `http://metalab.unc.edu/xml/`

[14] IBM Alphaworks. *A prototype Scalable Vector Graphics (SVG) viewer.* `http://www.alphaworks.ibm.com/tech/svgview`

[15] The Internet Society. Tim Berners-Lee *et al. Uniform Resource Identifiers (URI): Generic Syntax.* `http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2396.txt`

[16] International Organization for Standardization. *Information Technology – Processing Languages – Document Style Semantics and Specification Language (DSSSL). First edition, 1996* International Standard ISO/IEC 10179:1996, ISO Geneva, 1996.

A PDF version is at the URL `ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/dsssl96b.pdf`.

[17] International Organization for Standardization. *Information technology – Hypermedia/Time-based Structuring Language (HyTime)Hypermedia/Time-based Structuring Language (Hytime)* International Standard ISO/IEC 10744:1997, ISO Geneva, 1997.

[18] David Megginson. *SAX 2.0: The Simple API for XML* `http://www.megginson.com/SAX/`

[19] David Raggett. *Clean up your Web pages with HTML TIDY* `http://www.w3.org/People/Raggett/tidy/`

[20] Sebastian Rahtz. *Passive TEX* `http://users.ox.ac.uk/~rahtz/passivetex/`

[21] Sebastian Rahtz. *TEI and XSL.* `http://users.ox.ac.uk/~rahtz/tei/`

[22] Gaspar Sinai and Roman Czyborra. *Yudit Unicode editor.* `http://czyborra.com/yudit/`

[23] SISSA, Udine, City University, UJF, CERN, IOPP. *Tools for Innovative Publishing in Science (TIPS).* `http://jhep.sissa.it/proposal/`

[24] The Unicode Consortium. *The Unicode Standard, Version 3.0*. Addison-Wesley, Reading, 2000. See also `http://www.unicode.org`.

[25] Norman Walsh and Leonard Muelner. *Docbook. The Definitive Guide.* O'Reilly & Associates, Inc., Sebastopol, USA, 1999. `http://nwalsh.com/docbook/index.html`

The URL `http://www.oasis-open.org/docbook/documentation/reference/index.html` gives access to the reference documentation, while DTDs and stylesheets are at `http://nwalsh.com/docbook/index.html`

[26] E. Whitehead, UC Irvine, and M. Murata *XML Media Types*. `http://www.ietf.org/rfc/rfc2376.txt`

[27] World Wide Web Consortium. Håkon Wium Lie, Bert Bos, Chris Lilley and Ian Jacobs (editors). *Cascading Style Sheets, level 2.* `http://www.w3.org/TR/REC-CSS2`

[28] World Wide Web Consortium. Lauren Wood et al. *Document Object Model (DOM) Level 1 Specification. Version 1.0.* `http://www.w3.org/TR/REC-DOM-Level-1/`

[29] World Wide Web Consortium. Lauren Wood et al. *Document Object Model (DOM) Level 2 Specification. Version 1.0 (candidate recommendation)* `http://www.w3.org/TR/REC-CSS2`

[30] World Wide Web Consortium. *HyperText Markup Language (Home page).* `http://www.w3.org/MarkUp/`

[31] World Wide Web Consortium. Patrick Ion and Robert Miner (editors). *Mathematical Markup Language (MathML[tm]) 1.01 Specification.* `http://www.w3.org/TR/REC-MathML/`

[32] World Wide Web Consortium. Tim Bray, Dave Hollander and Andrew Layman (editors). *Namespaces in XML.* `http://www.w3.org/TR/REC-xml-names`

[33] World Wide Web Consortium. *QL'98 - The Query Languages Workshop.* `http://www.w3.org/TandS/QL/QL98/Overview.html`

[34] World Wide Web Consortium, David C. Fallside (editor). *XML Schema Part 0: Primer (W3C Working Draft).* `http://www.w3.org/TR/xmlschema-0`

[35] World Wide Web Consortium, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn (editors). *XML Schema Part 1: Structures (W3C Working Draft).* http://www.w3.org/TR/xmlschema-1

[36] World Wide Web Consortium, Paul V. Biron, Ashok Malhotra (editors). *XML Schema Part 2: Datatypes (W3C Working Draft).* http://www.w3.org/TR/xmlschema-2

[37] World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification (W3C Recommendation 15-June-1998)* http://www.w3.org/TR/REC-smil

[38] World Wide Web Consortium, Jon Ferraiolo (editor). *Scalable Vector Graphics (SVG) 1.0 Specification (W3C Working Draft).* http://www.w3.org/TR/SVG

[39] World Wide Web Consortium. *XHTML 1.0: The Extensible HyperText Markup Language. A Reformulation of HTML 4 in XML 1.0.* http://www.w3.org/TR/xhtml1/

[40] World Wide Web Consortium. Murray Altheim and Shane McCarron (editors). *XHTML 1.1 - Module-based XHTML.* http://www.w3.org/TR/xhtml11/

[41] World Wide Web Consortium. Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron and Ted Wugofski (editors). *Modularization of XHTML.* http://www.w3.org/TR/xhtml-modularization/

[42] World Wide Web Consortium, Steve DeRose, Ron Daniel Jr., and Eve Maler (editors). *XML Pointer Language (XPointer), W3C Working Draft 6 December 1999.* http://www.w3.org/TR/xptr

[43] World Wide Web Consortium. Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen (editors). *Extensible Markup Language (XML) 1.0.* http://www.w3.org/TR/REC-xml An annotated version of the specification is at http://www.xml.com/axml/axml.html.

[44] World Wide Web Consortium. Mary Fernandez and Jonathan Robie (editors). *XML Query Data Model (W3C Working Draft 11 May 2000)* http://www.w3.org/TR/query-datamodel/

[45] World Wide Web Consortium. Peter Fankhauser Massimo Marchiori, and Jonathan Robie (editors). *XML Query Requirements (W3C Working Draft 31 January 2000)* http://www.w3.org/TR/xmlquery-req/

[46] World Wide Web Consortium, James Clark and Steve DeRose (editors). *XML Path Language (XPath), Version 1.0 (W3C recommendation).* http://www.w3.org/TR/xpath

[47] World Wide Web Consortium, Steve DeRose, Ron Daniel Jr., and Eve Maler (editors). *XML Pointer Language (XPointer), W3C Working Draft 6 December 1999.* http://www.w3.org/TR/xptr

[48] World Wide Web Consortium, James Clark (editor). *XSL Transformations (XSLT), Version 1.0 (W3C Recommendation 16 November 1999).* http://www.w3.org/TR/xslt

[49] World Wide Web Consortium, Stephen Deach (editor). *Extensible Stylesheet Language (XSL), Version 1.0 (W3C Working Draft).* http://www.w3.org/TR/WD-xsl