

# **XML: A Technical Introduction.**

Massimo Martinelli  
Istituto di Elaborazione della Informazione (IEI)  
Consiglio Nazionale delle Ricerche (CNR)  
Via Alfieri, 1 - 56100 Pisa Italy  
email: M.Martinelli@iei.pi.cnr.it

## **Abstract**

This report aims at providing a technical introduction to the eXtensible Markup Language, known as XML. This language has been developed by W3C, the World Wide Web Consortium in order to offer an effective mechanism for managing structured data on the Web, thus overcoming the limits of HTML (Hyper Text Markup Language). We examine the components of the language and the way in which it differs with HTML. Examples are given of current possible uses and suggestion are made for future applications. We also briefly outline some of the new standards that are emerged, or now emerging with respect to XML.

## **Introduction**

The number of people accessing the Internet is growing as is the quantity of information published on the Web. The need to represent and use more complex information or information deriving from new technologies and new media, and also the need to manage and retrieve such information, is leading to the study of new languages that can be used for this purpose.

The eXtensible Markup Language, abbreviated XML, has been studied to use structured data on the Web and to overcome the limits of HTML (Hyper Text Markup Language).

XML is developed by W3C, the World Wide Web Consortium (the developers of HTML) and is a subset of SGML (Standard Generalized Markup Language), an international standard used to define the rules to write markup languages. XML does not include some of the complex functionality of SGML, considered difficult to implement on the Web.

## **XML specifications**

The first draft of XML was released in November 1996, the current specification is consultable at this address:

<http://www.w3.org/TR/1998/REC-xml-19980210>

## **The limits of HTML**

Why do we need a new language ? Let us examine some limits of HTML:

- it is not extensible, personal tags are ignored; to extend it a new version is needed;
- it cannot handle structured data;
- it is very difficult to retrieve data written in HTML; this can be done only using complex procedures built by experts in other programming languages, and the results are often not reliable (e.g. see the heterogeneous results we obtain from search engines);
- it is not a good exchange format;
- it is not dynamic;
- we cannot use it to process complex data;
- it says nothing about the data, it does not explain the data it contains.

The above problems provide an adequate answer to those who think that it is sufficient just to improve HTML. HTML has structural limits; we need a conceptually different language, similar to SGML but easier to use and more flexible.

However, it is clear that HTML will not be replaced, at least in the immediate future, because it provides the simplest method to publish information on the Web.

## **The goals of XML**

The goals of XML, as specified by W3C, are:

- 1) XML shall be straightforwardly usable over the Internet
- 2) XML shall support a wide variety of applications
- 3) XML shall be compatible with SGML
- 4) It shall be easy to write programs which process XML documents
- 5) The number of optional features in XML is to be kept to the absolute minimum, ideally zero
- 6) XML documents should be human-legible and reasonably clear
- 7) The XML design should be prepared quickly
- 8) The XML design should be formal and concise
- 9) XML documents shall be easy to create
- 10) Terseness in XML markup is of minimal importance

## Markup, Extensibility

Markup is all that has a special meaning, that must be well characterised: bold text, underlined text are examples of markup.

An XML document is composed of markups and text: markups represent the logical structure of the document, text is the content of the document, the data.

In XML all that is included between angled brackets ("`<`" and "`>`"), is considered markup, and is called tag, for example:

`<name>` is a tag.

XML is a metalanguage: Unlike HTML, which is a predefined language, it does not have predefined tags but allows the definition of new tags and new languages (there is an HTML version in XML). It is extensible.

HTML is also a markup language, it was initially defined in SGML. The set of HTML rules are contained in a document (generally included in the browser) called DTD HTML (*Document Type Definition*).

While the predefined tags of HTML are used to specify the visual aspects of the document, in XML they are used to structure the document, to define the content and to describe the data.

## The components of XML

One of the most common problems today is the exchange of documents: each program stores its data in one or more proprietary formats difficult to exchange with other programs.

XML has been studied to allow and facilitate data exchange between different kinds of applications, for example databases and word processors. The interest aroused by the new language is such that many software producers now intend to adopt the XML format or are already using it in their programmes.

For a document to be easy to interpret there must be three distinct parts:

- the content;
- the specification of the elements, the structure (DTD or Schema);
- the specification of the visual aspects, the style (Stylesheet).

Why is it so important to maintain these three components distinct? If we have to look for information in a book, we consult the index, when this is available, otherwise we check the chapters to see whether the information contained is inherent, we then scan the paragraphs, finally we look at the content.

A particular indication such as an indentation or the use of a different character helps us to understand where a chapter or a paragraph begins: the style should help us, to understand when reading the relation of the structure with the content.

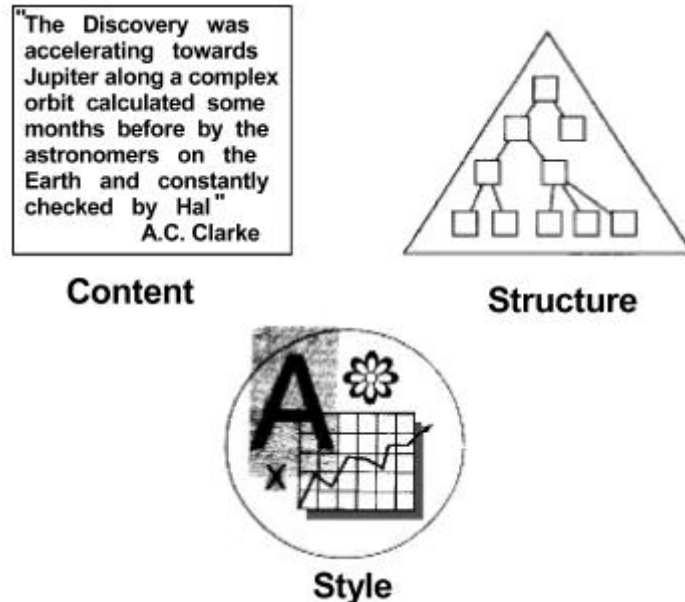
Without this distinction, an author concentrating on the visual properties risks losing sight of the content of his/her document.

With this distinction effort can be shared: the author, who has no need to know informatics elements, can concentrate on the content of his document entrusting the visual aspect to the graphic. In addition, it is more simple to write applications that must elaborate data. Once the content is separated

from the style, it is easier to integrate data from different sources and different stylesheet can be applied if needed.

If the language is modified and we want others know this or we want to use it as an exchange format, we need an open solution not a proprietary one. This means that the meaning of the extensions made must be declared, with a public DTD.

**Figure 1 – the three components of a document**



### **An example of an XML document**

According to the design goals of XML, it should be human legible. It must be readable by any text reader, such as Unix vi or Windows Notepad. It should not be in binary format (even if P. Hoschka, W3C member, during the inauguration of the W3C office in Italy talked about the possibility of a binary XML), and should be reasonably clear. The following example, library.xml, shows how an XML document can be written:

**Figure 2 - library.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <book code="R414">
    <title>2001: A Space Odissey</title>
    <author>
      <first_name>Arthur Charles</first_name>
      <last_name>Clarke</last_name>
    </author>
    <publisher>Polaris Production</publisher>
    <keyword>romance</keyword >
    <keyword>science-fiction</keyword >
  </book>
</library>
```

The document begins with a prologue that contains a declaration of conformity to version 1.0 of the XML standard and to the UTF-8 encoding standard:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Elements are used to declare the associated content and are written in the form:

```
<element_type_name attribute_name="attribute_value">element content</element_type_name>
```

The content of an element can be of the following types: character data, parsed character data (character data that can be evaluated by parsers, programs capable of reading and interpreting XML documents), processing instructions (information to give to programs) or other nested elements.

An element can have attributes, used to better specify the content.

Each open tag must have a corresponding final tag; when the element is empty, like the case of BR in HTML, instead of <BR></BR> the use of the concise form <BR/> is permissible.

XML is case sensitive, so a <name> tag is different from <Name> and from <NAME>.

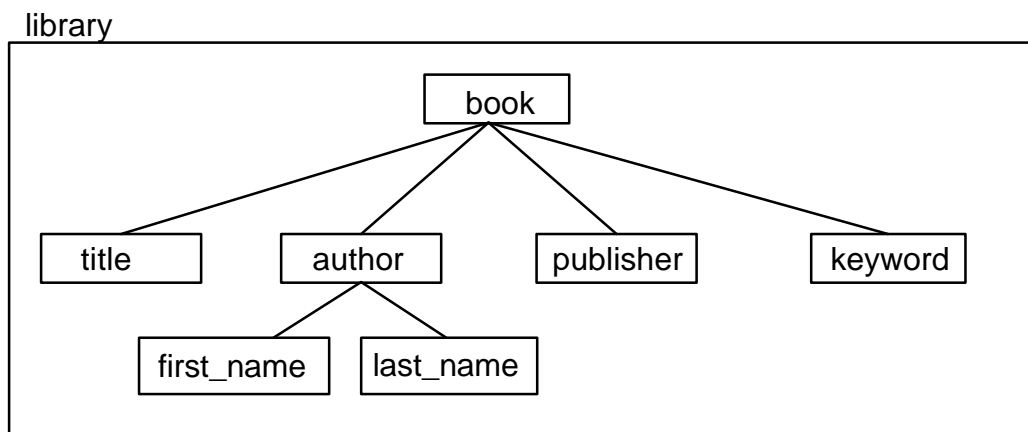
Some characters and sequences of characters are reserved and cannot be used in the names of the tags (% , xml , ...).

XML supports Unicode (Unicode WorldWide Character Standard), a standard code system that supports characters of the diverse languages of the world, both modern and also historical languages. With Unicode, browsers should automatically render the right character even if a document contains words written in several code sets.

## **DTD - Document Type Definition**

The DTD contains the definition rules of tags; it denotes the elements and their order inside the XML document. Unlike SGML, its use is not compulsory, but is suggested in order to verify the validity and congruence of the document. Other than defining the elements, it defines the syntax and the relations among elements.

The structure of an XML document is seen as a tree in which elements represent the nodes. The following figure (Fig. 3), represents the structure of a hypothetical library:



**Figure 3 – The tree structure**

The unique element that contains all the others is called the root element. The DTD can be internal or external to the XML document. By convention its name corresponds to that of the root element (in our example "library").

The following figure (Figure 4) shows an XML DTD, library.dtd, representing the previous defined structure:

**Figure 4 - library.dtd**

```
<!DOCTYPE library [  
  <!ELEMENT library (book+)>  
  <!ELEMENT book (title, author+, publisher, keyword+)>  
  <!ATTLIST book  
    code ID #REQUIRED>  
  <!ELEMENT title (#PCDATA)>  
  <!ELEMENT author (first_name, last_name)>  
  <!ELEMENT publisher (#PCDATA)>  
  <!ELEMENT keyword (#PCDATA)>  
  <!ELEMENT last_name (#PCDATA)>  
  <!ELEMENT first_name (#PCDATA)>  

```

A DTD can be internal or external to the XML document; generally it is written in one or more separate documents.

If internal to the XML document, the DTD starts with a DOCTYPE declaration:

```
<!DOCTYPE root_element [element, attribute, entity, notation]>
```

otherwise this declaration must be made inside the XML document; the library.xml file would be written:

```
<?xml version="1.0"?>  
<!DOCTYPE library SYSTEM "library.dtd">  
<library>  
  ...  
</library>
```

After the DOCTYPE declaration we have to declare elements:

```
<!ELEMENT element_name (permitted_elements_names)>
```

Examining the declaration of the "book" element, we can note that is composed by the elements "title", "author", "publisher" and "keyword". These elements are separated by a comma and thus must be present in that exact order in the library.xml file.

Let us examine the following declaration:

```
<!ELEMENT person (first_name|last_name)>
```

```
<!ELEMENT person (first_name|last_name|email)*>
```

In the first case the "|" character means that the "person" element will be constituted by the "first\_name" or by the "last\_name" element; in the second

case, in which a "\*" character is present, by 0, 1 or more "first\_name", "last\_name", "email" elements in any order.

We can see that the author element is followed by the "+" character: this means that there must be one or more "author" elements.

If an element, or a group of elements listed between parentheses, is followed by the "?" character it can be present none or one times.

<!ELEMENT title (#PCDATA)> means that the title element can be composed by any character except for markup strings or special characters like ", & or ] (PCDATA = *Parsed Character Data*).

Attributes are used generally as specifications of elements, to add information to elements. In this case, we can consider attributes as metadata, i.e. information about information. But attributes are also used to have a control on values; we can oblige data to be a value of a predefined list, like the enumerated type in the C language.

Attributes are declared in the form:

```
<!ATTRIBUTE element_name
attribute_name attribute_type default_value>
```

In our example, each book element will have a univocal code (ID).

Attribute types can be CDATA (character data), ID, or a list of values (enumerated type) and are declared in this way:

```
<!ATTLIST element_name
attribute_name attribute_type default_value>
```

The attribute value is compulsory when the option #REQUIRED is specified (in our example the code of the book element is compulsory), it can have no specified code if the option is #IMPLIED, or only the specified value is valid with the option #FIXED.

In addition to elements and attributes, the DTD can contain entities and notations.

An XML document can be composed by elements referencing other objects; these objects are called entities. The following example shows a document internal entity:

```
<!ENTITY XML "eXtensible Markup Language">
```

In this case, during visualisation, each &XML occurrence will be replaced by the string "eXtensible Markup Language".

Entities can be used to represent reserved characters like, for example, "<" or ">" (lt, &gt;), but can also be used to reference external objects, such as other XML documents or images, when the document does not consist of a unique file. The following example defines an external entity:

```
<!ENTITY introduction SYSTEM "introduction.txt">
```

A document can also contain parametrical entities:

```
<!ENTITY % [name] "[a_list_of_names]">
```

for example:

```
<!ENTITY % headings "H1 | H2 | H3 | H4">
<!ENTITY BODY (%headings | P | DIV | BR)*>
```

The notation declaration is useful to identify specific types of external binary data, like for example files in "

```
<!NOTATION GIF87A SYSTEM "GIF">
```

In case we need to preserve the space contained in an element, for example in a poem, we use the `xml:space` attribute set to "preserve"

```
<!ELEMENT poem (#PCDATA)>
<!ATTLIST poem
    xml:space (default| preserve) "preserve">
```

Processing instructions used to pass information to the applications are written in the form "`<?name PIdata?>`" where name, called the Ptarget, is used to identify the process instruction to the programs.

The DTD is required when we have to define default attribute values, when we need to handle white spaces and when other programs need to read it to understand the structure of the document.

In order for a document to be considered "well formed":

- it must contain at least one element;
- there must be a unique open and close tag that contains the entire document; this tag is called the root element;
- all the tags must be nested (unlike HTML `<B><I>text</B></I>` is not allowed);
- all the entities must be declared.

A document is said "valid" when it is associated with a DTD and respects the rules defined in the DTD.

## Namespaces

An XML document may have content relative to more than one argument or discipline, or collect data from different documents or resources; this can be a source of problems because there could be several elements or attributes in the same document with the same names but relative to different contexts. Let's consider, for example, the word "mouse": in one case it can be referred to a pointing device, in another to an animal or to a counter-balance or to a shy person; consider also "tree" (as a structure, or a plant).

To eliminate possible ambiguities, namespaces have been introduced. As defined by W3C, a namespace is a collection of names, identified by a URI, which are used in XML documents as element types and attribute names. These multiple collections are also called vocabularies or markup vocabularies. A name is constituted by two parts: the local name or the name of element type name as has been shown until now, a prefix which specifies the name of the namespace and a colon dividing the two parts. The namespace must be declared using an attribute with the prefix "xmlns" and must have a unique URI.

In the following example

```
<html:body xmlns:html="http://www.w3.org/TR/REC-html40">
  <html:h1>text</html:h1>
</html:body>
```

the body and h1 elements are mapped to the space defined by the HTML 4.0 specifications as follows:

```
<{http://www.w3.org/TR/REC-html40}body>
  <{http://www.w3.org/TR/REC-html40}h1>text</{http://www.w3.org/TR/REC-html40}h1>
</{http://www.w3.org/TR/REC-html40}body>
```

### Some other examples:

<pre>&lt;s xmlns="http://www.a.it"&gt; ... &lt;/s&gt;</pre>	maps to	<pre>&lt;{http://www.a.it}s&gt;   all the childs of s are mapped   with the same namespace of s   if is not specified other &lt;/{http://www.a.it}s&gt;</pre>
<pre>&lt;a xmlns="http://www.1"   xmlns:i="http://www.2"&gt;   &lt;b&gt; &lt;/b&gt;   &lt;i:c&gt; &lt;/i:c&gt;   &lt;d xmlns=""&gt;     &lt;e&gt; &lt;/e&gt;   &lt;/d&gt;   &lt;f xmlns="http://www.3"&gt;     &lt;g&gt; &lt;/g&gt;   &lt;/f&gt;   &lt;h&gt; &lt;/h&gt; &lt;/a&gt;</pre>	maps to	<pre>&lt;{http://www.1}a&gt;   &lt;{http://www.1}b&gt; &lt;/{http://www.1}b&gt;   &lt;{http://www.2}c&gt; &lt;/{http://www.2}c&gt;   &lt;d&gt;     &lt;e&gt; &lt;/e&gt;   &lt;/d&gt;   &lt;{http://www.3} &gt;     &lt;{http://www.3}g&gt; &lt;/{http://www.3}g&gt;   &lt;/{http://www.3}f&gt;   &lt;{http://www.1}h&gt; &lt;/{http://www.1}h&gt; &lt;/{http://www.1}a&gt;</pre>
<pre>&lt;a xmlns:n="http://www..."&gt;   &lt;b n:c="value"&gt; &lt;/b&gt;   &lt;n:d e="value"&gt; &lt;/n:d&gt; &lt;/a&gt;</pre>	maps to	<pre>&lt;a&gt;   &lt;b {http://www...}c="value"&gt; &lt;/b&gt;   &lt;{http://www...}d e="value"&gt; &lt;/{http://www...}d&gt; &lt;/a&gt;</pre>

The W3C recommendation for namespaces is consultable at:  
<http://www.w3.org/TR/1999/REC-xml-names-1990114>  
 RFC 2396 about URI can be found at:  
<http://www.ietf.org/rfc/rfc2396.txt>

## The schema

We have described the namespaces and you will have probably noticed that they are not supported by DTDs. DTDs have other great limitations:

- they are not written in XML;
- they are difficult to write;
- they are not extensible;
- they do not specify datatypes (or, more precisely, they specify only one type of data, the parsed character data) and very often applications need to check for specified data.

A solution has been proposed to extend the functionality of DTDs; this is known as the schema.

A schema provides an extensible way to define the XML data model: in addition to defining elements, attributes and their relationships as a DTD, it provides the possibility to add information, such as datatypes and inheritance. Specifying datatypes makes it possible to constrain rules, such as applying ranges to values, specifying the length of a string, or the decimal precision of a number.

The following example shows a schema for our document library.xml, written in the xml-data, a format proposed by Microsoft:

```
<Schema name="library" xmlns="urn:schemas-microsoft.com:xml-data"
        xmlns:dt="urn:schemas-microsoft.com:datatypes">
  <AttributeType name="code" dt:type="id" required="yes" />
  <ElementType name="title"          content="textOnly" dt:type="string" />
  <ElementType name="first_name"     content="textOnly" dt:type="string" />
  <ElementType name="last_name"      content="textOnly" dt:type="string" />
  <ElementType name="publisher"      content="textOnly" dt:type="string" />
  <ElementType name="keyword"        content="textOnly" dt:type="string" />
  <ElementType name="author" content="eltOnly" model="closed" order="seq">
    <element type="first_name" minOccurs="1" maxOccurs="1" />
    <element type="last_name"  minOccurs="1" maxOccurs="1" />
  </ElementType>
  <ElementType name="book" content="eltOnly" model="closed" order="seq">
    <attribute type="code" />
    <element type="title"      minOccurs="1" maxOccurs="1" />
    <element type="author"     minOccurs="1"
    <element type="publisher" minOccurs="1" maxOccurs="1" />
    <element type="keyword"   minOccurs="1"
  </ElementType>
  <ElementType name="library" >
    <element type="book" minOccurs="1" maxOccurs="*" />
  </ElementType>
```

A new Working Draft, XML-Schema, was released in March; this tries to merge the different solutions proposed (XML-Data, DCD, SOX, DDML, XDR) into a common one that provides domains and methods of application. It is divided into two parts, Structures and Datatypes.

The datatypes provided are of two kinds: primitive and derived. Primitive are built-in ready to use objects, like numbers, logical values (boolean) or characters. Derived datatypes are objects that anyone can derive from the built-in ones using methods provided by the schema language.

The following is an example of xml-schema for our library.xml document:

```
<?xml version=1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/1999/XMLSchema"
  targetNamespace=" http://www.my.org/mySchema"
  version="1.0" >

  and the target namespace is my own namespace -->

  <!-- we define a new datatype, of simple type, named "
    whose base datatype is "string", the max length of an object
    of this type must be of 60 characters -->
  <xs:simpleType name="titleType" base="string">
    <xs:maxlength value="60" />
  </xs:simpleType>

  <xs:simpleType name="publisherType" base="string">
    <xs:maxlength value="30" />
  </xs:simpleType>
  <xs:simpleType name="nameType" base="string">
    <xs:maxlength value="30" />
  </xs:simpleType>

  <xs:simpleType name="keywordType" base="string">
    <xs:maxlength value="10" />
  </xs:simpleType>

  <!-- we define a complex datatype named "authorType"
    that is constituted by two elements called first_name and
    last_name of "nameType" type that must occur only once
    The content of a complex type could be empty, or could contain
    a simple type definition or a pair consisting of a content model
    (sequence, all, choice) and one of mixed element only.
    There are three types of structure:
    - sequence: must contain the elements listed in the exact order
    - all: must contain the elements listed in any order
    - choice: must contain one of the elements listed
    With mixed keyword we can insert any text inside
    the structure -->
  <xs:complexType name="authorType">
    <xs:element name="first_name" type="nameType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="last_name" type="nameType" minOccurs="1" maxOccurs="1"/>
  </xs:complexType>

  <!-- we define a new simple datatype of "ID" base type that must be
    <!-- we define a new simple datatype of "ID" base type that must be
    formed by 3 characters in the set of letters from "a" to "z",
    lower or upper case, followed by a "\" character and then by 3 digits
  -->
```

```

<xs:simpleType name="codeType" base="ID">
  <xs:pattern value="[a-zA-Z]{1}\d{3}" />
</xs:simpleType>

<!-- the complexType we now define contains a required attribute
      and defines a structure of "sequence" type -->
<xs:complexType name="bookType">
  <xs:attribute name="code" type="codeType" minOccurs="1" />
  <xs:sequence>
    <xs:element name="title" type="titleType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="author" type="authorType" minOccurs="1" maxOccurs="*/>
    <xs:element name="publisher" type="publisherType"
      minOccurs="1" maxOccurs="1"/>
    <xs:element name="keyword" type="keywordType" minOccurs="1" maxOccurs="*/>
  </xs:sequence>
</xs:complexType>

<!-- we declare an element "library". This element contains an
      anonymous (unnamed) complexType that contains an element "book"
      that must occur at least once and can occur more than once -->
<xs:element name="library">
  <xs:complexType>
    <xs:element name="book" type="bookType" minOccurs="1" maxOccurs="*/>
  </xs:complexType>
</xs:element>

</xs:schema>

```

An XML document that conforms to a schema is called a document instance. If we use a schema instead of a DTD in addition to the structure of a document a validating parser must check also that the data types are in conformity with the declared ones, plus bounds and patterns. It is probable that parsers will implement three functionality: non validating, structure validating and validating both the structure and the datatypes. We will use one of these depending on the context.

In the previous example we have used only one base datatype, but many others are permitted as types using numbers, types for text, types for date and time. For a complete list see the Working Drafts.

The XML-Schema Working Drafts are consultable at:  
<http://www.w3.org/2000/05/06-xmlschema-0>  
<http://www.w3.org/2000/05/06-xmlschema-1>  
<http://www.w3.org/2000/05/06-xmlschema-2>

## Stylesheet

With the style separated from the content and from the structure it is no longer necessary to rewrite the entire document when we change the presentation. We only need to rewrite the elements that are necessary for the formatting. Instead of referring to a specific visualisation on the web, the stylesheet will be modified to send the output to other devices, for example to a speech synthesiser, to A3 format paper for printing, or simply for a presentation on the web in a different way, or with different elements. In this way a document associated with a stylesheet today will be reusable on a new device tomorrow.

As the XML specification does not refer to any particular style, to render the document on the web we can use CSS (*Cascade Style Sheet*), to transform the document output into HTML or to use proprietary formats that are applicable only on specific platforms and by specified programs can be used.

The HTML browsers interpret tags for visualisation in their own way or may even use proprietary tags, that involve different outputs. Microsoft Internet Explorer 5 contains a default stylesheet that makes it possible to display an XML document.

Even if the last version of CSS (specifications can be found at <http://www.w3.org/TR/REC-CSS2>) has added new possibilities, the current stylesheet is limited in its potential as a result of the already mentioned inherent limitations of HTML.

Figures 5 and 6 show a CSS stylesheet for library.xml, library.css, and the visualisation of the library.xml document with Internet Explorer 5:

**Figure 5 - library.css**

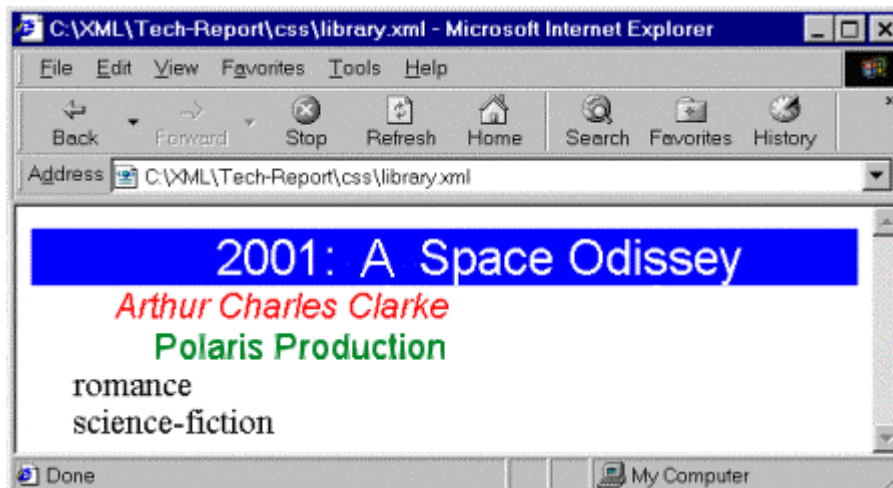
```
title { display: block;
        text-align: center;
        background: blue;
        color: white;
        font-family: Arial;
        font-size: 20pt
    }
author { display: block;
        margin-left: 10%;
        text-align: left;
        color: red;
        font-family: Arial;
        font-style: italic;
        font-size: 14pt
    }
first_name, last_name { display: inline;
    }
publisher { display: block;
        margin-left: 15%;
        color: green;
```

```

        font-family: Arial;
        font-size: 14pt
    }
keyword { display: block;
        margin-left: 5%;
        color: black;
        font-family: "Times New Roman";
        text-align: justify
        font-size: 14pt
    }

```

If we exclude "display:block" (line feed and carriage return) and "display:inline" (continue on the same line), the previous example is almost self-explanatory. We can see that the visualisation properties are set to obtain only a static representation of the original objects of the document, shown in the following figure:



**Figure 6 – Viewing library.xml with CSS style**

To overcome the limits of HTML and CSS, new styles are now being studied for XML. In particular XSL (*eXtensible Stylesheet Language*) is a style language based on DSSSL (*Document Style Semantics and Specification Language*); this is used in particular way for SGML documents.

XSL is divided into three parts: a language for referencing specific parts of an XML document (XML Path language – XPath), a language for transforming XML documents into other XML documents (eXtensible Stylesheet Language Transformations-XSLT) and an XML vocabulary for specifying formatting semantics. XSL reads the XML file and then constructs a tree of the objects of the document. It transforms the original tree into a result tree applying transformation rules and then formats this for the media to be employed (display, printer or other devices, devices allowable in future are not excluded). The transformation could produce a result tree that is completely different from the original one: objects can be added, sorted and filtered.

Formatting is obtained through the object formatting semantics that defines the visual and physical aspects of the object (Es: page and font format).

Let us see an example of XSL in use.

**Figure 7 - library.xsl**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
                xmlns:html="http://www.w3.org/TR/REC-html40"
                result-ns="">

<xsl:template match="/">
<HTML>
  <HEAD> <TITLE> Library </TITLE> </HEAD>
  <BODY background="Paper.jpg">
    <FONT face="Arial" ><H1><CENTER><b>Library</b></CENTER></H1></FONT>

    <xsl:for-each select="library/book">
      <HR/>
      <P>Book code <xsl:value-of select="./@code"/></P>
      <xsl:apply-templates />
    </xsl:for-each>
  </BODY>
</HTML>
</xsl:template>

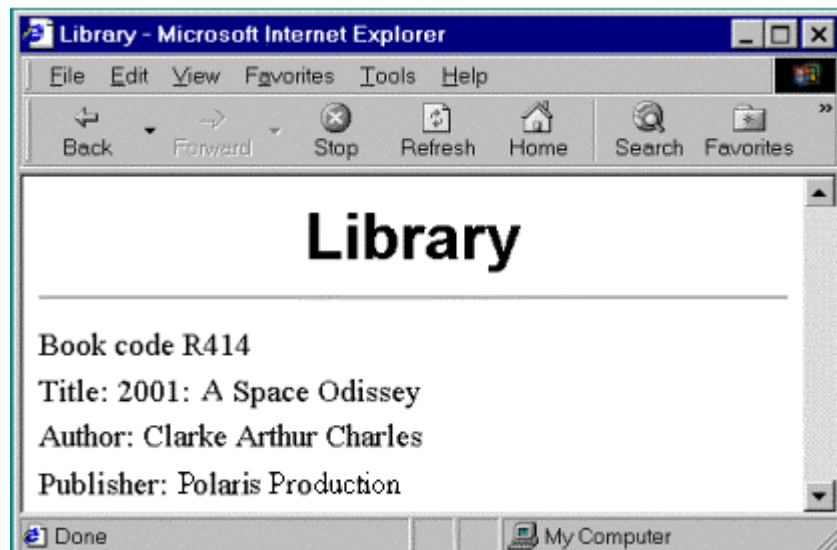
<xsl:template match="title">
  <P>Title: <xsl:value-of select="."/></P>
</xsl:template>

<xsl:template match="publisher">
  <P>Publisher: <xsl:value-of select="."/></P>
</xsl:template>

<xsl:template match="author">
  <P>Author: <xsl:value-of select="./last_name" />
    <xsl:value-of select="./first_name" /></P>
</xsl:template>

</xsl:stylesheet>
```

**Figure 8 - A view with an XSL stylesheet**



An XSL stylesheet is itself an xml document. In the example, after the usual prologue of declarations we see the declaration of two namespaces: elements with the prefix xsl are style elements as defined at <http://www.w3.org/TR/WD-XSL> (the Working Draft of XSL); elements with the prefix html are style elements as defined at <http://www.w3.org/TR/REC-html40> (HTML 4.0 style elements). The `result-ns=""` statement says that all the elements without prefix will be mapped to the last namespace declared. In this case it is HTML.

A series of template rules follows. When the "pattern" of a template rule matches an object of the source tree, a new object is constructed; this is called flow object, as specified by the "action" part of the template. At the end of this process, the result tree is passed to the user agent.

```
<xsl:template match="/">
```

searches for the root of the document and then applies the following transformations/formattings

```
<xsl:for-each select='library/book' >
```

is a recursive rule that tells the parser to select all the books child of library

```
<xsl:value-of select='./@code' />
```

shows the value of the attribute code of the current node (book), `./` is used to reference a relative location path

```
<xsl:apply-templates />
```

applies the templates to the book objects

The for-each rule tells the parser to apply the other rules of the xsl file relative to the objects contained in the book objects. In this case those for title, publisher and author.

```
<xsl:template match="title">
```

```
<P>Title: <xsl:value-of select="." /></P>
```

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

shows the text "Title:" and then the value of the current node (.) i.e. the title.

Let us see other examples:

```
<xsl:template match="library">
```

```
<xsl:apply-templates select="book[code >= 'M']" >
```

```
<xsl:sort select="book/author/last_name" />
```

```
<xsl:sort select="book//fist_name" />
```

```
</xsl:apply-templates>
```

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

In this case the parser searches for the library node, then selects the book nodes whose code is greater than or equal to "M", sorts the nodes first by last\_name and then by first\_name, and finally applies the templates.

Note other relative location paths: book/author/last\_name or last\_name child of author child of book, book//first\_name or first\_name with a book ancestor.

Supposing we have more than one author and we want to divide them by a comma, we could use the following template:

```
<xsl:template match="author">
  <xsl:value-of select="./last_name" />
  <xsl:value-of select="./first_name" />
  <xsl:apply-templates />
  <xsl:if test="not(position()=last())">,</xsl:if>
</xsl:template>
```

in this way we have a comma after all the authors with the exception of the last one.

Here we have a more complex form of conditional processing:

```
<xsl:template match="element">
  <xsl:variable name="var"
    expr="subelement[position() mod 3]"/>
  <xsl:choose>
    <xsl:when test='$var=1'>
      ... do something ...
    </xsl:when>
    <xsl:when test='$var=2'>
      ...do something different ...
    </xsl:when>
    <xsl:otherwise>
      ... do something else ...
    </xsl:otherwise>
  </xsl:choose>
  <xsl:apply-templates/>
</xsl:template>
```

Note the declaration of a variable whose value is a mathematical expression.

The following example shows a simple XSL stylesheet with formatting objects (FO). These are used to specify the representation of the result tree constructed with XSLT (the rendering has been obtained using the Indelv Browser).

**Figure 9 - an XSL stylesheet with formatting objects**

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:fo="http://www.w3.org/TR/WD-xsl/FO"
  result-ns="fo">

<xsl:template match="/">
  <fo:page-sequence
    font-family="Times New Roman, Serif"
    font-size="12pt">
  <xsl:apply-templates />
  </fo:page-sequence>
</xsl:template>

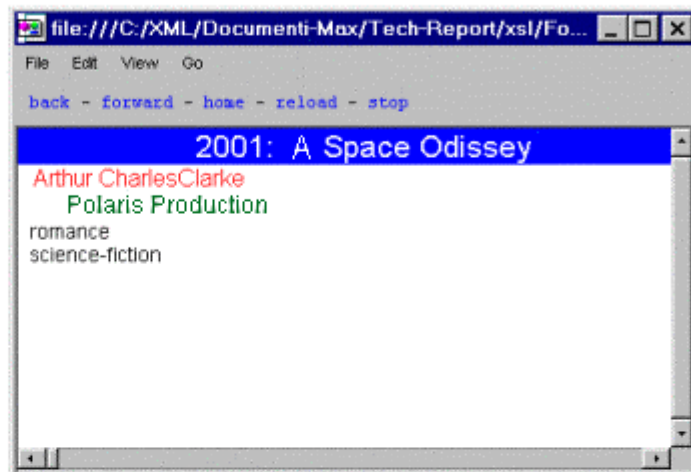
<xsl:template match="title">
  <fo:block
    background-color="blue"
    color="white"
    font-family="Verdana"
    font-size="18pt"
    font-weight="bold"
    text-align="centered">
  <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="author">
  <fo:block
    color="red"
    font-family="Arial"
    font-size="14pt"
    start-indent=' 6pt'
    end-indent=' 6pt'>
  <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="publisher">
  <fo:block
    color="green"
    font-family="Arial"
    font-size="14pt"
    start-indent=' 20pt'
    end-indent=' 20pt'>
  <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="keyword">
  <fo:block
    color="black"
    font-family="Arial"
    font-size="12pt"
    start-indent=' 5pt'
    end-indent=' 5pt'>
  <xsl:apply-templates />
  </fo:block>
</xsl:template>

</xsl:stylesheet>
```



The Working Draft of XSL is available at <http://www.w3.org/TR/2000/WD-xsl-20000327>  
the recommendation of XPath at <http://www.w3.org/TR/xpath>

To associate stylesheets with an XML document we need to insert a processing instruction, after the prologue:

```
<?xml-stylesheet href="library.xml" type="text/xsl" ?>
```

the attributes href and type are obligatory but they have following also been defined:

```
title CDATA #IMPLIED  
media CDATA #IMPLIED  
charset CDATA #IMPLIED  
alternate (yes|no) "no"
```

alternate="yes" can propose an alternative stylesheet.

The recommendation "Associating Style Sheets with XML documents is downloadable at the address:

<http://www.w3.org/TR/1999/06/REC-xml-stylesheet-1990629>

While XSLT is a W3C recommendation (<http://www.w3.org/TR/1999/REC-xslt-19991116>), XSL is still a Working Draft: the potentiality and simplicity of XSL, greater than CSS, clearly suggest that it will become the reference style for XML.

## XML Linking Language

One of the basic functions that have encouraged the growth of Internet and, in particular, of hypertexts is the link.

SGML does not support markup links; HTML currently only implements a unidirectional link.

It is under developing a language, XLL (*eXtensible Linking Language*) that should provides XML with links that are not only of the unidirectional type. XLL is developing in two directions: Xlink and Xpointer.

Xlink (*XML Link Language*) allows links between more than two resources. These links can be used to indicate whether the linked resource must be inserted in a precise point of the current document (embed - in HTML this was possible only with frames), whether should replace the current document (replace), or whether a new window must be opened (new). It is possible to use bi-directional links when the information flows are activated between two resources, or with unique links when several destination (multiple link) are indicated. The link can be activated automatically (auto) or manually (user - usually when the user clicks on the text or on the image associated with the link).

The W3C document draft on XLink is consultable at:

<http://www.w3.org/TR/2000/WD-xlink-20000221>

XPointer (*XML Pointer Language*) is a language that specifies locations.

It makes possible to link to a point inside a document even if this has not been referenced. In HTML this is possible only if the author of the document has inserted an anchor, with XPointer it is always possible because it is easy to locate any point in a document through its structure or to refer to it, depending on the context. For example, the fourth element of the first "name" child of the document can be referred through the following notation:  
`child(1,name).child(4,#element)`

The working draft of Xpointer is allowable at:

<http://www.w3.org/TR/1999/WD-xptr-1991206>

If the linking element itself is one of the resources participating in the link we have a inline link, otherwise the link is out-of-line.

Until now, if we had two targets to link together we had to create a link for each of them. Now, just one link is sufficient.

Let us see the example:

```
<P NAME="target1"> <A HREF="#target2">a link to target2</A></P>
```

```
<P NAME="target2"> <A HREF="#target1">a link to target1</A></P>
```

The relation can now be created elsewhere, also in an external document and only once (not for each resource). This remains transparent to the user.

Let us see another example:

```
<myns:myelement xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="#target1"
  xlink:href="#target2" >
</myns:myelement>
...
<name ID="target1"> .... </name>
...
<title ID="target2"> .... </title>
```

In this case, the link can also be called bidirectional: from either of the two points we can link to the other (this is different from a backward pointing link. It is also out-of-line, because the linking element is not a resource participating in the link.

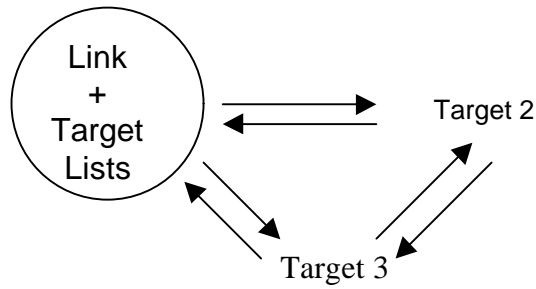
In the following example the "myelement" element participates in the link, and we have an extended inline link.

```
<myns:myelement id="target1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="extended"
  xlink:href="#target2"
  xlink:href="#target3" >
</myns:myelement>
...
<name ID="target2"> .... </name>
...
<title ID="target3"> .... </title>
```

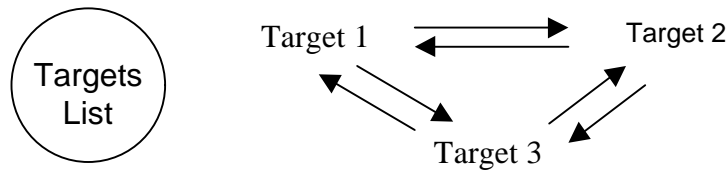
A link is simple if it occurs only between two resources; such links are said to be unidirectional and usually they are inline. When a link occurs between two or more resources, it can be multidirectional; such links can be inline or out-of-line, they enable database linking.

Figures 10 and 11 should help to understand the inline and outline links.

**Figure 10 – An Inline Link**



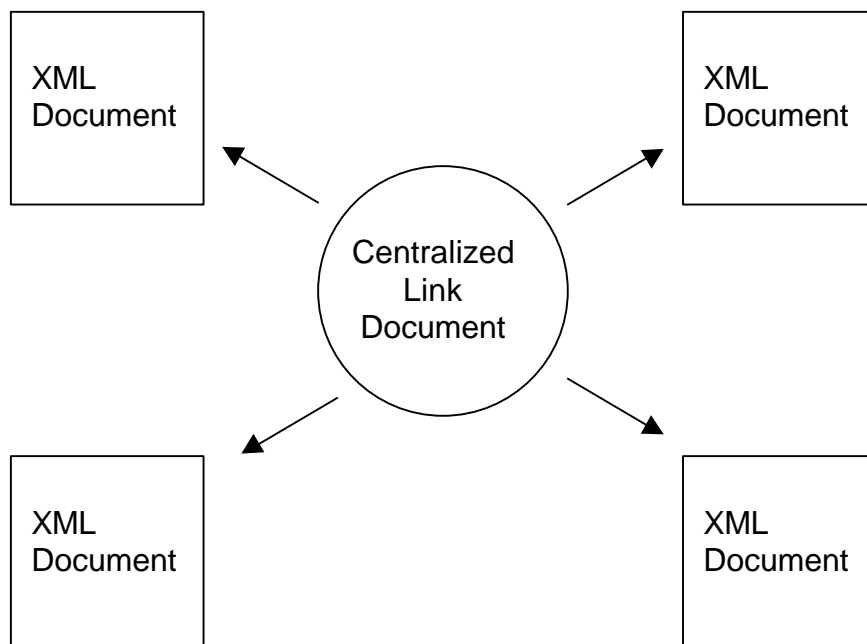
**Figure 11 – An Out-Of-Line Link**



These link types make it possible to simplify things when we have many targets to link together and we have not to create a number of links equivalent to the number of the targets minus one for each target but we have to create only one time a list of targets occurring.

This is very important when a correlated resource changes, instead of changing the link in each resource in which it occurs, with the risk of the "broken links" phenomenon (links pointing to targets that no longer exist), we can now create a database document in which to locate the links: when we change a resource we only need to check this document to modify links.

**Figura 12 – A Centralized Link Document**



This method can be used in all kinds of documents which must be related to other resources, such as Stylesheets, Schema or Scripts.

Other attributes could be used to pass information to the user agent, which must be capable of interpreting the values, as "role" used to explain the meaning of the link, or "title" to give a title of the link. Let us look at the example

```
<myns:myelement>
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="index.html"
  xlink:role="index"
  xlink:title="Table of contents"
  xlink:show="new"
  xlink:actuate="onRequest">
Table of contents
</myns:myelement>
```

In the following example the traversal (the action of using a link), is not multidirectional but directional: we specify the starting and the terminating resource, the inverse direction is not allowed. This specification of link traversal rules is called arc.

```
<directional xlink:from="#target1" xlink:to="#target2"/>
```

Another example (taken from the working draft), is an extended directional link:

```
<extendedlink>
  <loc xlink:role:"parent" xlink:title="p1"/>
  <loc xlink:role:"parent" xlink:title="p2"/>
  <loc xlink:role:"child" xlink:title="c1"/>
  <loc xlink:role:"child" xlink:title="c2"/>
  <loc xlink:role:"child" xlink:title="c3"/>
</extendedlink >
.....
<go xlink:from="parent" xlink:to="child"/>
```

In this case six directional links have been specified from each parent to each child (p1->c1, p1->c2, p1->c3 and p2->c1, p2->c2, p2->c3)

Extended and out-of-line links offers enhanced capabilities but are more complex to manage: browsers, or more generally applications processing these kind of documents, must first look for links (that may be not specified in-line). Because links can be in different documents, we need to tell the browser where it must look for them. A previous working draft proposed a "group" element with a "steps" attribute; in the most recent one the application is required to limit the number of steps to be processed.

Look at the following example:

```
<externallinkset>
  <linkbase xlink:href="http://linkset1.xml"/>
  <linkbase xlink:href="http://linkset2.xml"/>
  <linkbase xlink:href="http://linkset3.xml"/>
</externallinkset>
```

The "externallinkset" element specifies which documents must be checked by the XML application for links. The processor must check the three XML documents linkset1, 2 and 3 that could contain links to other external links.

## DOM - Document Object Model

Suppose you have a document with the following content:

The data you have requested:

The first day the values of A, B, C were 5,7,Normal  
those of D,E,F respectively 55, 20, Normal.

The following day values were 6, 8, Abnormal and 40, 18, Abnormal.

As humans we can identify immediately the values we need from this document, however is not so simple to write a program to extract the required data and any program will function only for the specified document.

The program we have just created will function only with that specified document. We can simplify our life as programmers rewriting the variables and the values between markups, in the following way:

```
<data>
  <day value="1">
    <a>5</a><b>7</b><c>Normal</c>
    <d>55</d><e>20</e><f>Normal</f>
  </day>
  <day value="2">
    <a>6</a><b>8</b><c>Abormal</c>
    <d>40</d><e>18</e><f>Abormal</f>
  </day>
</data>
```

Now we can define a set of functions that we can reuse with any other document that associates specific markups with the elements. For example, we can write a function that, given the name of an element, returns its value.

The aim of the DOM (*Document Object Model*) is to define a platform and a language-independent interface that allows programs to dynamically access and modify the content, the structure and the style of documents. The actual recommendation (level 1) shows how to access and modify the content of the document. This model can be applied both to XML and to HTML documents.

Level 1 of DOM is divided into two parts: Core and HTML.

The first part defines a set of low-level interfaces to represent structured documents, the core, and higher level interfaces to represent XML documents. The second part defines a set of interfaces, using those defined in the Core, for a simple access to HTML documents.

The DOM offers instruments to query on elements and attributes and a model to manage events that could be generated by any element.

A mechanism will be provided for error management, in addition to the security mechanisms that will evolve with the levels of the DOM.

The level 1 specifications are consultable at the following address:

<http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001>

Through the DOM, the elements of the XML document are seen as nodes of a tree (nodes represent not a structure but objects) or more precisely of a forest

that can contain more than one tree. The DOM thus provides methods to manipulate these objects in this form.

These are the types of nodes we can find:

	Associated value (W3C)	Associated value (Microsoft)
Document	1	9
Element	2	1
Attribute	3	2
ProcessingInstruction	4	7
Comment	5	8
Text	6	3
CDATASection	7	4
DocumentFragment	8	11
Entity	9	6
EntityReference	10	5
DocumentType	11	10
Notation	12	12

We can see that the value associated with the types of node that Microsoft uses in its Internet Explorer differs from the W3C recommendation.

Each node can be seen as an object that can have some associated methods. Let us see some methods offered by the DOM:

```
Node.getDocumentElement()    it returns the root element of the document
Node.getFirstChild()        it returns the first child of a given node
Node.getLastChild()         it returns the last child of a given node
Node.getNextSibling()       it returns the next sibling a given node
Node.getPreviousSibling()   it returns the previous sibling a given node
```

Supposing we have to add a new element to our document library.xml:

```
X=document.createElement(newelement);
```

creates a new element called "newelement"

```
Document.documentElement().book[0].appendChild(x);
```

inserts as a new child of the first element book that is child of root (document.documentElement)

```
x=createTextNode("content of the new element");
```

creates a new text type node

```
newelement[0].appendChild(x);
```

assigns it to newelement

## Let's see now a practical example

```
<HTML>
<HEAD>
  <TITLE>test</TITLE>
</HEAD>
<BODY>
  <XML ID="island" SRC="library.xml"></XML>
<SCRIPT>
  // creates an XML document object
  var myDoc=island;

  // checks for validity
  if(myDoc.parseError.reason != "")
  {
    alert(myDoc.parseError.reason);
  }
  else
  {
    alert("Valid XML document");
  }

  // assigns the root element to the variable rootEl
  rootEl=myDoc.documentElement;
  // assign book to curEl
  curEl=rootEl.firstChild;
  // creates a new attribute "position" and assigns it to book
  newAttr=curEl.setAttribute("position","F3-51");

  // assigns keyword(0) to the variable curEl
  curEl=rootEl.firstChild.childNodes.item(3);
  newEl=myDoc.createElement("pages");
  rootEl.firstChild.insertBefore(newEl,curEl);
  // creates a text node with the value of 215
  txtNd=myDoc.createTextNode("215");
  // creates a list of elements whose tag name corresponds to pages
  pages=myDoc.getElementsByTagName("pages")
  pages(0).appendChild(txtNd);

  // book
  curEl=rootEl.firstChild;

  // creates a list of attributes;
  attlist=curEl.attributes;

  // assigns to id the value B509
  attlist.item(0).nodeValue="B509";
  // it is equal to attlist.namedItem("id").nodeValue="B509"

  document.write("<TABLE BORDER='1'>");
  document.write("<TR>");
  document.write("<TD>Node Name</TD>");
  document.write("<TD>Type</TD>");
  document.write("<TD>Value</TD>");
  document.write("</TR>");
  // loop through it's content
```

```

x=printchildren(myDoc);

// ----- begin recursive function 'printchildren' -----

function printchildren(node)
{
  // creates a child list
  var x=node.childNodes;
  var z=x.length;
  if(z!=0)
  {
    for(var i=0; i < z; i++)
    {
      document.write("<TR>");
      document.write("<TD>" + x(i).nodeName + "</TD>");
      document.write("<TD>" + x(i).nodeType + "</TD>");
      document.write("<TD>" + x(i).nodeValue + "</TD>");
      document.write("</TR>");

      printchildren(x(i));
    }
  }
}

// ----- end function -----

document.write("</TABLE>");

// creates a list of attributes of book
attrL=rootEl.firstChild.attributes;
document.write("The value of " + attrL.item(0).nodeName + " is " +
attrL.item(0).nodeValue);

</SCRIPT>
</BODY>
</HTML>

```

The XML island way has been used to create an XML document object through an "XML" tag, as proposed in a document of Frank Boumphrey available at this address:

<http://www.w3.org/MarkUp/future/papers/boumphrey-19980419.html>

"parseError" is a specific method of Internet Explorer, it does not form part of the DOM.

Browsing the above document, we first see a window stating that the XML document is valid, as shown in Figure 13A. If the document is not valid, the reason will be given.

We then see the results, as shown in Figure 13B.

**Figure 13A – Information about validity**



test - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search

Address [C:\XML\Tech-Report\DOM\DOM.html](file:///C:/XML/Tech-Report/DOM/DOM.html)

Node Name	Type	Value
xml	7	version="1.0" encoding="UTF-8"
library	10	null
library	1	null
book	1	null
title	1	null
#text	3	2001: A Space Odyssey
author	1	null
first_name	1	null
#text	3	Arthur Charles
last_name	1	null
#text	3	Clarke
publisher	1	null
#text	3	Polaris Production
pages	1	null
#text	3	215
keyword	1	null
#text	3	romance
keyword	1	null
#text	3	science-fiction

The value of code is B509

Done My Computer

## Some applications

New languages defined with XML are called applications: even if XML has only been in existence for a few months there are already many applications.

Some examples:

*Mathematical Markup Language* (MathML) defines a language for mathematics, *Chemical Markup Language* (CML) defines a language for chemistry, *Channel Definition Format* (CDF) is used as open format to exchange information on channels, *Resource Definition Format* (RDF) is an instrument to describe metadata, *Open Software Description* (OSD) is used to describe software, *Synchronized Multimedia Integration Language* (SMIL) is used to describe multimedia elements, *Scalable Vector Graphics* (SVG) is used for Vector Graphics.

The above listed applications are only some of the public standardised but extensible DTDs; for the most part they are derived from DTDs used in SGML. New applications will soon be available for many disciplines.

## The software

A program to process XML documents is constituted by a "parser" (analyser) and by a browser. The parser checks whether the document is in conformity with the DTD (if the document is "well formed") and, if validating, its validity ("valid" document). The browser visualises the document on the basis of the stylesheet.

The following table lists some relevant programs:

Parsers:	MSXML (Microsoft), XML for Java (IBM), XJParser (DataChannel), Oracle, Sun, ...
Parser Interfaces:	Sax, DOM written in Java, Perl, Python, Frontier, C++, ...
Editors:	Xmetal(Softquad), FrameMaker-SGML (Adobe), Adept (ArborText), Balise (Chrystal), XML-Authority, ...
Databases:	Oracle, DB2 (announced), Tamino, Excelon, ...
Browsers:	IE5.*, Netscape 6 beta, Indelv, Opera, Jumbo (usato per CML), Amaya (usato per MathML), DocZilla...

## Where to find Other Information

As XML is a new language and the related standards are in continuous evolution, books in commerce risk already being outdated at the moment of publication. The best way to find updated documentation is on the web. The following list of addresses constitutes a good starting point:

The home page of XML on the W3C site: <http://www.w3.org/XML>

The FAQ distributed by some of the XML Working Group members:

<http://www.ucc.ie/xml>

The page of SGML/XML group: <http://www.oasis-open.org/cover/sgml-xml.html>

Microsoft's XML site: <http://www.microsoft.com/xml>

Netscape's site: <http://www.mozilla.org>

The GCA page (Graphics Communication Association) on XML:

[http://www.gca.org/conf/xml/xml\\_what.htm](http://www.gca.org/conf/xml/xml_what.htm)

The ArborText site: <http://www.arbortext.com/xml.html>

The university of XML: <http://www.xmlu.com>

The site of XML Italia group , which also maintains an XML mailing-list:

<http://www.xml.it>      <http://listserv.xml.it/xml.html>

## Conclusions

The main problems to be resolved when surfing the Web are the difficulty of discovering the information we really need and also its slowness. These problems are both caused by the limitations of HTML.

A language is needed that defines the structure of the information contained in documents and makes it possible to define their content.

This language should be independent of the hardware and software platforms, guaranteeing a reusability that in this moment does not exist.

Devices connected to the Internet can currently receive and send documents to a Web server. More advanced operations are difficult because it is arduous to extrapolate and process data.

XML offers instruments to define the content of a document. XML also makes it possible to structure documents and to associate a syntax with them.

In the opinion of its developers, this means that XML will lead to a reduction in the network traffic, allowing a greater development of client applications. The server will send only relevant data to the client that will process it as required.

Using HTML, a document may be displayed in different ways, depending on the browser used, but it may be impossible to interpret it if it is written in a foreign language. However, XML supports UNICODE, so that a large number of character sets can be used.

XML can be used as exchange format between databases, word processors, spreadsheets and other applications. It may well become the most important exchange format, a universal standardised extensible exchange format for electronic documents and applications.

By expliciting structure and using an extensible tag-based language, as XNK does, it is possible to have a simple interaction with other programs, including data bases, and thus a simpler and more efficient handling of the data. In many cases, XML can replace existing applications.

When we store information, we want to be sure that it will be reusable in future. Using a proprietary format we are obliged to use the specific product that has created it and no other (except for a format converter). If, for example, we take a file produced by a word processor and try to read it with a normal text reader we will have problems. On the other hand an XML document is human-legible and self-explanatory.

Giving a meaning to a document means that we can build more accurate search engines returning more accurate results.

This capability to give information to information is leading to an increasing interest in the study of standard metadata systems: the Resource Description Framework is one of the most promising standard for the better and faster cataloguing and retrieving of documents.

With XML, we are no longer obliged to use a predefined set of tags: XML is extensible; allows new markups to be added, has no predefined tags.

Similar concepts will soon be applicable to datatypes: the schema is released with predefined datatypes but we can create new types that extend or restrict the characteristics of the built-in ones.

It is simpler for a programmer to write software that finds the markup and uses it. He can exploit the DOM that provides a base library of functions with a language and platform independent interface.

Links allows new possibilities, objects can be inserted in precise points, there be many destinations, but they are more manageable. They can be collected in a centralised database from where they can be easily changed when a web page is changed and checks link consistency becomes simple.

With XML, a new language has been defined that is useful to structure documents and to give metainformation. However, a language that stipulates how the information contained in the XML documents must be displayed is still missing.

Once the style language becomes a standard we will be able to render a document on different media or in different ways without rewriting it each time. The standard part of this language offers a good mechanism of representation and a very good capability to represent complex data (mathematical notations, graphical interfaces). It makes it possible to effect cycles and to impose conditions: procedures that are at the base of a real programming language.

This report described many of the functionality offered by XML. The aim has been to show the advantages of using XML, and to outline possible future developments.

## Bibliography

### Technical Reports

- 1) "Extensible Markup Language (XML)" - *Tim Bray, Jean Paoli, C.M. Sperberg-McQueen*. <http://www.w3.org/XML>
- 2) "XML Schema Part 0: Primer" - *David C. Fallside*.  
<http://www.w3.org/TR/xmlschema-0>
- 3) "XML Schema Part 1: Structures" - *Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn*.  
<http://www.w3.org/TR/xmlschema-1>
- 4) "XML Schema Part 2: Datatypes" - *Paul V. Biron, Ashok Malhotra*.  
<http://www.w3.org/TR/xmlschema-2>
- 5) "HTML 4.01 Specification" - *Dave Ragget, Arnaud LeHors, Ian Jacobs*.  
<http://www.w3.org/TR/html401>
- 6) "XHTML 1.0: The Extensible HyperText Markup Language – A Reformulation of HTML 4 in XML 1.0" - *Dave Ragget, Arnaud LeHors, Ian Jacobs*.  
<http://www.w3.org/TR/xhtml1>
- 7) "Cascading Style Sheets, level 2 (CSS2)" - *Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs*. <http://www.w3.org/TR/REC-CSS2>
- 8) Extensible Stylesheet Language (XSL) - *Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, Steve Zilles*. <http://www.w3.org/TR/xsl>
- 9) XSL Transformations (XSLT) Version 1.0 - *James Clark*.  
<http://www.w3.org/TR/xslt>
- 10) XML Path Language (XPath) Version 1.0 - *James Clark, Steve De Rose*.  
<http://www.w3.org/TR/xpath>
- 11) "Document Content Description for XML (DCD)" - *Tim Bray, Charles Frankston, Ashok Malhotra*. <http://www.w3.org/TR/NOTE-dcd>
- 12) Document Description Markup Language (DDML )  
<http://www.w3.org/TR/NOTE-ddml>
- 13) Schema for Object-Oriented XML (SOX) - *Matt Fuchs, Murray Maloney, Alex Milowski*. <http://www.w3.org/TR/NOTE-SOX>
- 14) "XML-QL: A Query Language for XML" - *Alin Deutch, Mary Fernandez, Daniela Floescu, Alon Levy, Dan Suciu*. <http://www.w3.org/TR/1998/NOTE-xml-ql-19980819>
- 15) "Mathematical Markup Language (Math-ML) 1.01 Specification" - *Patrick Ion, Robert Miner, Stephen Buswell, Nico Poppelier*.  
<http://www.w3.org/TR/REC-MathML>
- 16) "Document Object Model (DOM) Level 1" - *Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud LeHors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, Lauren Wood*.  
<http://www.w3.org/TR/REC-DOM-Level-1>
- 17) XML Linking Language (Xlink) - *Steve DeRose, David Orchard, Ben Trafford*. <http://www.w3.org/TR/WD-xlink>
- 18) Xlink principles - *Eve Maler, Steve DeRose*.  
<http://www.w3.org/TR/NOTE-xml-link-principles>
- 19) "XML Pointer Language (XPointer)" - *Steve DeRose, Ron Daniel*.  
<http://www.w3.org/TR/WD-xptr>

- 20) "Resource Description Framework (RDF) Schemas" – *Dan Brickley, R.V. Guha*. <http://www.w3.org/TR/rdf-schema>
- 21) "XML Path Language (XPath)" - *James Clark, Steve DeRose*.  
<http://www.w3.org/TR/xpath>
- 22) "XML Query Language (XQL)" *Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, Dan Suciu*. <http://www.w3.org/TR/NOTE-xml-ql>
- 23) "Namespaces in XML" – *Tim Bray, Dave Hollander, Andrew Layman*.  
<http://www.w3.org/TR/REC-xml-namespaces>
- 24) "Using XSL and CSS together" – *Håkon Lie, Bert Bos*.  
<http://www.w3.org/TR/NOTE-XSL-and-CSS>

### Articles/Tutorials/Books

- 1) XML faq: <http://www.ucc.ie/xml/faq.txt>
- 2) XML faq Italian translation: <http://ada2.unipv.it/%7esimoz/FAQ-XML-1999216.htm>
- 3) Unicode <http://www.unicode.org> -  
[ftp://ftp.unicode.de/Public/UNIDATA/Unicode Data latest.txt](ftp://ftp.unicode.de/Public/UNIDATA/Unicode%20Data%20latest.txt)
- 4) DOM FAQ <http://www.w3.org/DOM/faq.html>
- 5) "A Technical Introduction to XML" - *Norman Walsh*. <http://www.xml.com>
- 6) "XML Tutorial" - *Frank Boumphrey*.  
<http://www.hypermedic.com/style/xml/index.html>
- 7) "XML Tutorials for Programmers"  
<http://www.software.ibm.com/xml/education/tutorial-prog/abstract.html>
- 8) "Introducing the Extensible Markup Language (XML)" - *Robin Cover*.  
<http://www.oasis-open.org/cover/xmlIntro.html>
- 9) "XML: Structuring Data for the Web" - *Ken Sall*.  
<http://www.wdvl.com/Authoring/Languages/XML/Intro/index.html>
- 10) "XML for Dummies" - *Ed Tittel, Norbert Mikula, Ramesh Chandak*.  
Apogeo.
- 11) IDG Books Worldwide
- 12) "XML Applications" *Frank Boumphrey, Olivia Drenzo, Jon Duckett, Joe Graf, Paul Houle, Dave Hollander, Trevor Jenkins, Peter Jones, Adrian Kingsley-Hughes, Kathy Kingsley-Hughes, Craig MacQueen, Stephen Mohr*.  
Wrox Press Ltd. <http://www.wrox.com>
- 13) "La seconda rivoluzione" Internet News Luglio Agosto 1998
- 14) "Introduction to XML" <http://msdn.microsoft.com/xml>
- 15) "XML Namespaces" *James Clark*. <http://www.jclark.com/xml/xmlns.htm>
- 16) "XML-Data" – *Andrew Layman, Edward Jung, Eve Maler, Henry S. Thompson, Jean Paoli, John Tigue, Norbert H. Mikula, Steve DeRose* –  
<http://www.w3.org/TR/1998/NOTE-XML-data>
- 17) "XML and the Second-Generation Web" - *Jon Bosak, Tim Bray*.  
Scientific American May 1999  
<http://www.sciam.com/1999/0599issue/0599bosak.html>
- 18) "XSL Concepts and Practical Use" – *Norman Walsh, Paul Grosso*. XML '99  
Philadelphia <http://www.nwalsh.com/docs/tutorials/xml99/xsl>

## Software

- 1) Amaya. <http://www.w3.org/Amaya>
- 2) Mozilla. <http://www.mozilla.org>
- 3) Netscape. <http://www.netscape.com>
- 4) DocZilla. <http://www.doczilla.com>
- 5) Oasis-Open. <http://www.oasis-open.org>
- 6) ArborText. <http://www.arbortext.com>
- 7) SAX. The Simple API for XML. <http://www.megginson.com/SAX>
- 8) Roland Bourret's XML Database Products. <http://ww.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLDatabaseProds.htm>
- 9) XML Software. <http://www.xmlsoftware.com>