Introduction to XML and its applications

Chapter · January 2013

DOI: 10.1142/9789812836304_0006

1 author:

Laura Papaleo
Rensselaer Polytechnic Institute
48 PUBLICATIONS **562** CITATIONS

SEE PROFILE

CHAPTER

INTRODUCTION TO XML AND ITS APPLICATIONS

Laura Papaleo

Department of Informatics and Computer Science, University of Genova
Via Dodecaneso, 35 16100 Genova, Italy
E-mail: papaleo@disi.unige.it – laura.papaleo@gmail.com

Extensible Markup Language (XML) is a meta-language for defining new languages. Its impact on the modern and emerging web technologies has been (and will be) incredible and it has represented the foundation of a multitude of applications. This chapter is devoted to the presentation of XML and its applications. It provides an introduction to this wide topic, covering the principal arguments and providing references and examples.

1. Introduction

XML is hugely important. It has been defined as "*the holy grail of computing, solving the problem of universal data interchange between dissimilar systems*" (Dr. Charles Goldfarb). XML is basically a handy format for everything from configuration files to data and documents of almost any type. The first version of XML became a W3C Recommendation in 1998, while its fifth edition has been declared recommendation last year, in 2008 [1].

XML is significant, but it is a hard subject to describe briefly in a chapter, because it describes a whole family of technologies and specifications. In 10 years, its success has been incredible and it has represented the foundation of a multitude of applications.

This chapter has the goal to present the XML meta-language, trying to give an overview of the most significant parts. We will describe the syntax to create XML documents and how we can structure them by defining specific grammars (DTDs and XML Schemas). We will also show how to render XML documents using CSS style sheets and how to transform and render them with a family of XML-based languages (XSL, XSLT and XPath). The end of the chapter will be dedicated to provide a snapshot of the "life" around XML, to let the reader understand the immense impact of XML in the actual technological world.

2. What Extensible Markup Language (XML) is

Extensible Markup Language (XML) [1] is a simple, very flexible text format used for the description of marked-up electronic content. XML is classified as *extensible* because it allows the user to define the mark-up elements [2]. It is defined as a *markup language* because it allows to make explicit an interpretation of a text using a set of markup conventions (tags).

More exactly, XML is a *meta-language*, that is, a means of formally describing a language, in this case, a markup language. Today, XML is playing an important role in the exchange of a wide variety of data on the Web and elsewhere [1]. Generally speaking, XML's purpose is to aid information systems in sharing structured data, especially via the Internet, to encode documents, and to serialize data. XML, in combination with other standards, makes it possible to define the content of a document separately from its formatting, making it easy to *reuse* that content [2]. Most importantly, XML provides a basic syntax that can be used to *share* information between different applications and different organizations without using expensive and time-consuming conversion [3].

3. Origins of the Extensible Markup Language

XML emerged as a way to overcome the shortcomings of its two predecessors, the *Standard Generalized Markup Language* (ISO 8879:1986 SGML) and the *HyperText Markup Language* (HTML) which are both restricted in some ways. Roughly speaking, HTML is too limited, while SGML is too complex. XML, instead, is a software- and hardware-independent light and simple tool for carrying information [4,5].

The key point is that using XML, scientific organizations, industries and other companies, can specify how to store specific data in a machine-understandable form so that applications - running on any platform - can easily import and process these data. In the following subsections we will briefly present SGML and HTML and we will outline their main differences with respect to XML. This will help the reader to understand why XML has been defined. Finally, we will shortly recall the history of the XML birth.

3.1 *Standardized Generalized Markup language - SGML*

The Standardized Generalized Markup Language (SGML, for short) - conceived notionally in the 1960s - 1970s, has been considered, since its beginning, the international standard for marking up data and it is an ISO

standard since 1986, ISO 8879:1986. SGML has been defined as a powerful and extensible markup language with the main goal to semantic markup any type of content. This functionality is particularly useful for cataloging and indexing data [7]. SGML can be used to create an infinite number of markup languages and has a host of other resources as well.

Historically it has been used by experts and scientific communities. However, SGML is really complex and expensive: adding SGML capability to an application could double its price. Thus, from the Web point of view, the commercial browsers decided not to support SGML.

Both SGML and XML are widely-used for the definition of device-independent, system-independent methods of storing and processing texts in electronic form. Comparing the two languages, basically, XML is a simplification or derivation of SGML, developed thinking at the emerging Web technologies [7].

3.2 *Hypertext Markup Language - HTML*

A well known application of SGML is HTML (Hypertext Markup Language), which is the publishing language of the World Wide Web. HTML defines a specific (finite) set of tags for structuring content and for publishing it over internet. HTML is free, simple and widely supported. Thousand of online tutorials, books and web portals exist describing the HTML language. In this section we would like to concentrate on the main problems of the old versions of HTML, which have had a certain significance in pushing the formalization of XML.

HTML has serious defects. The original thinking was to separate content from presentation, but the evolution of HTML lost this purpose (as for example due to the use of tags as `<em>` and `<font>`). Web pages began to be used for things that went wildly beyond the original concept, including multimedia (using the tag `<object>`), animation and many more. Thus, they started to become more containers of more fascinating objects (e.g. flash animations) than pages describing content, arising big problems in web searching performance. Also, browsers tried to be tolerant of incorrect web pages and this tolerance became a barrier to programmatic interpretation of published content, as for the use of HTML for structured data.

XML arose from the recognition that key components of the original web infrastructure - HTML tagging, simple hypertext linking, and hardcoded presentation - would not scale up to meet the future needs of the web [8].

Compared with HTML, XML has some important characteristics. First of all XML is *extensible* so it does not contain a fixed set of tags. Additionally, XML documents must be well-formed according to a strict set of rules, and may be formally validated (using DTDs or XML Schemas), while HTML documents can contain errors and still the browsers render the pages as well as possible. Also, XML focuses on the *meaning of data*, not its presentation.

It is important to understand that XML is not a replacement for HTML. In most web applications, *XML is used to transport data*, while HTML is used to format and display the data for the Web. Additionally, thanks to XML, HTML evolved into XHTML [9] which is basically a reformulation of HTML (version 4) in XML 1.0.

3.3 *The birth of XML*

In 1996, discussions began which focused on how to define a markup language with the power and extensibility of SGML but with the simplicity of HTML. The World Wide Web Consortium (W3C) founded a working group [9] on this goal, which came up with XML, the *eXtensible Markup Language*. Like SGML, XML had to be not itself a markup language, but a specification for defining markup languages. Like HTML, XML was planning to be very simple to learn and clear in the syntax.

Since the beginning, the design goals for XML were clear to the working group and they can be summarized in the following 10 points:

(i) XML shall be straightforwardly usable over the Internet.
(ii) XML shall support a wide variety of applications.
(iii) XML shall be compatible with SGML.
(iv) It shall be easy to write programs which process XML documents.
(v) The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
(vi) XML documents should be human-legible and reasonably clear.
(vii) The XML design should be prepared quickly.
(viii) The design of XML shall be formal and concise.
(ix) XML documents shall be easy to create.
(x) Terseness in XML markup is of minimal importance.

Over the next years, XML evolved: by mid 1997 The *eXtensible Linking Language XLL* project was underway and by the summer of 1997, Microsoft had launched the Channel Definition Format (CDF) as one of the first real-world applications of XML. Finally, in 1998, the W3C approved Version 1.0 of the XML specification as Recommendation. A new language was born reaching

completely the planned goals. In 2008, the fifth edition of XML has been approved as W3C recommendation and the working groups on XML are still active.

4. XML Documents: Syntax

As we have seen, XML is a formal specification for markup languages. Every formal language specification has an associated syntax. In this section, we will briefly recall the syntax of XML documents. More details and information can be found in the XML specifications from W3C [1] or in books as [3,4,6]

An XML document consists of a *prolog*, that includes an XML declaration and an optional reference to external structuring documents and the *body* consisting of a *number of elements* which may contain also *attributes*. These elements are organized in a hierarchical structure (a *tree*), meaning that there can only be one *root element*, also called the *Document element*, and all other elements lie within the root. In Fig. 1 we show an example of an XML document and the corresponding tree structure.

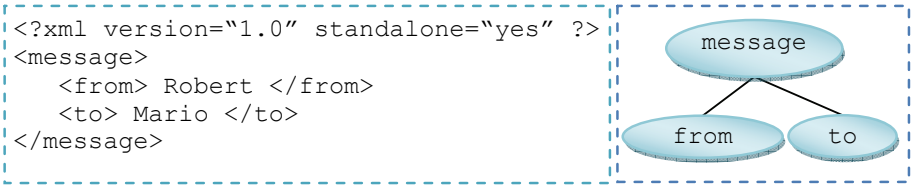


Fig. 1 – an example of XML document (left) and the associate tree structure (right)

The first line of the prolog is the *declaration* (see Fig. 2) and it serves to let the machine understanding that what follows is XML, plus additional information such as the encoding. Other components that can be inserted in the prolog of an XML document as, for example, the associated schemas (either DTDs or XML Schema – see Section 6 and Section 7) or the attached style sheets (in CSS or XSL – see Section 8 and 9).

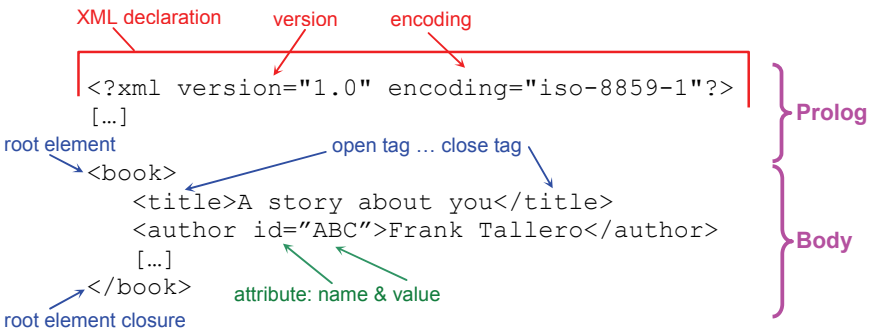


Fig. 2 – an example of XML document. The prolog and the body are outlined as the root element. The image also show the syntax for opening and closing a tag and the syntax for an attribute.

XML document *body* is made up of elements (see Fig. 2). Each *element* is defined using two basic components *data* and *markup*. Data represents the actual content, thus the information to be structured. Markup, instead, are *meta-information* about data that describes it. What follows is an example for an element, structuring the information regarding a message body:

```
<message> This is the content </message>
```

For a reader who is familiar with HTML, the XML elements will be easy to understand, since the syntax is very similar. The markup are tags in the form `<tagName>…</tagName>`. Elements can also contain other elements and can have attributes. For the attributes, again, the syntax is very simple. Each attribute can be specified only in the element start tag and it has a *name* and a *value* and the value is enclosed strictly in double quotation-mark. Fig. 2 shows an example of XML document outlining the prolog, the body, the elements and the attributes.

Empty elements do not have the closing tag `</tagName>`, instead, they have a “/” at the end. Code (4.1) represents an empty tag with attributes.

```
<book isbn=”A234DX” />
```
(4.1)

When an element contains additional elements or attributes, it is defined as *complex*. In the following we present two codes in XML of complex elements, structuring the same information:

```
<message>
  From Robert                                                    (4.2)
  <to>Mario</to>
</message>
```

```
<message from="Robert">                                          (4.3)
  <to>Robert</to>
</message>
```

In the years, different discussions are arisen within scientific and technical communities on when and why to encode information into attributes or as content in elements. There is not a specific rule and the choice depends on the designer. However, XML attributes are normally used to *describe* elements, or to provide additional information about elements. So, basically, *metadata* (data about data) should be stored as attributes, and that data itself should be stored as elements.

When necessary, comments can be inserted into an XML document. Their syntax is the same as for comments in HTML and it is the following:

```
<!-- This is a comment -->
```

An XML document can also contain *processing instructions*. Generally speaking, processing instructions encode application-specific data. The document declaration which opens every XML document is an example of a processing instruction. Processing instructions contain a target followed by data. Each instruction is enclosed in `<?` and `?>` delimiters. The target identifies the application, and an application should ignore processing instructions for targets it does not recognize. Basically, processing instructions allow to enter directives into a XML document which are not part of the actual content, but which are passed up to different ad-hoc applications.

An XML document can use also *entities*. It could be easier to think of entities as a macro for programmers, or as aliases for more complex functions. A single entity name can take the place of a whole lot of text. An example of entity is showed below:

```
<!ENTITY myname "Laura">
```

Once defined, an entity can be recalled in the content of the document using the syntax "`&myname;`" and the following is a piece of XML code showing how to use the entity `myname`.

```
<details> My name is &myname; </details>
```

Finally, in a XML document *CDATA elements* can be used. A CDATA element tells the XML parser not to interpret or parse characters that appear in the section. An example is the following, where the content is parsable, even though it contains an unparsable character (the ampersand):

```
<hobbies><![CDATA[Singing & Swimming]]></hobbies>
```

4.1 *Well-formed XML document*s

Unlike HTML, which allows to create documents with errors in the structure which will be still rendered in a browser, XML has strict rules and a XML document must be correctly structured in order to be machine-understandable. The XML specification prohibits XML parsers from trying to fix and understand malformed documents. All a conforming parser is allowed to do is report the error.

Thus, a XML document must be *well-formed*. According to W3C, a well-formed XML document is defined as a document that:

- has at least one element
- contains a unique opening and closing tag that enclose the whole document, called the root element
- has all the elements with the closing tag, or empty elements correctly written
- has all the tags and attributes names written accordingly to the case-sensitive rule, that is, for example that the tag `<name>` cannot be closed with `</Name>`. In other words, elements and attribute names may be any case chosen, as long as they are consistent.
- has all the elements properly nested, i.e. there must be an opening and a closing tag and the tags cannot overlap. For example if the tag `<name>` has been opened after the tag `<person>`, it must be closed before.
- has all the attribute values always quoted correctly.

These are the most important constraints for the well formdness, but they are far to be a complete list: the XML Specifications [1] provides all the necessary details.

Well formed XML documents simply markup content with descriptive tags. This means that there is not the necessity to describe or explain what the chosen tags mean. We will see in Section 6 and Section 7 how DTDs and XML Schemas can define the meaning of the tags and can force the structure.

5. Namespaces

In XML, element names are defined by developers. This means that different organizations can use the same tag to markup content with different semantics. But XML has been invented also to allow interoperability and data exchange among different organizations so there must exist a way to combine several XML sources without ambiguity.

XML namespaces are used for providing uniquely named elements and attributes in an XML instance [13]. They are defined by a W3C recommendation called Namespaces in XML. As defined by the W3C, an XML namespace is a collection of XML elements and attributes identified by an Internationalized Resource Identifier (IRI); this collection is often referred to as an XML *vocabulary* [14].

Using namespaces, name conflicts can be solved thus allowing the correct integration among data. This means that, if each vocabulary has given a namespace then the ambiguity between identically named elements or attributes can be resolved.

Namespaces are declared as an attribute of an element by using the `xmlns` name attribute in the start tag of the element. It is not mandatory to declare namespaces only at the root element; rather it could be declared at any element in the XML document. A namespace has a scope which begins at the element where it has been declared and applies to the entire content of that element, unless overridden by another namespace declaration with the same prefix name [15]. A namespace is declared as follows, that can be read as binding the prefix "*myname*" with the namespace *http://www.whatever.com*:

```
<someTag xmlns:myname="http://www.whatever.com" />
```

So, the namespace declaration has the following syntax. `xmlns:prefix="URI"`. A Uniform Resource Identifier (URI) [3,16] is a string of characters which identifies an Internet Resource. The most common URI is the Uniform Resource Locator (URL) which identifies an Internet domain address. Another, not so common type of URI is the Universal Resource Name (URN). Note that the URI is not actually read as an online address; it is simply treated by an XML parser as a string. Note also that, the empty string, though it is a legal URI reference, cannot be used as a namespace name.

A specific namespace identifies a *collection of names*. This collection can be made of element names, as in the case of the standard XHTML [9] or it can be a collection of attribute names, as in the case of XLink [17]. A namespace can

collects names of properties (e.g. FOAF [18]) or can describe a set of functions, as it is the case for the XPath 2.0 Data Model [19].

6. Structuring XML Documents: Document Type Definition

As we said before, using XML, any developer can create his own well-formed documents in freedom, without any restriction or specific template rules on how to organize the tags. But, in case of organizations in which the XML documents must follow a specific grammar to be sharable and usable (as in the case of technical reports, e-commerce transactions, workers details), tools for describing the *shape* of all specific-topic XML documents are necessary [20].

The purpose of DTDs is exactly this. They provide a framework for validating XML documents by defining the legal building blocks of XML documents [3]. Basically, a *Document Type Definition* (DTD) outlines what elements can be in an XML document and the attributes and sub-elements that they can take. Thus DTDs allow different organizations to create shareable data files.

A DTD can be part of the XML document, or it can be referred to by the XML document. In the first case, we call it an *inline* DTD while in the second case it is called *external* and it is a simple text file with "`.dtd`" extension.

DTDs embody a small syntax that can be mastered quite quickly. This syntax has several important components but can be summed into two essential structures, which are the *element* and the *attribute*. In the following subsections, we will describe the syntax used for these two type of declarations.

6.1 *Element Declaration*s

Element declarations describe the allowable set of elements within the document, and specify whether and how declared elements (and character data) may be contained within each element.

Recall that (Section 4) elements in XML documents can enclose other elements, can be empty, can contain content or can be mixed (containing content and other elements). In a DTD the possible declarations for elements are the following:

- `<!ELEMENT element-name (child1,child2,...)>` , for an element containing other elements
- `<!ELEMENT element-name EMPTY>`, for an empty element
- `<!ELEMENT element-name (#PCDATA)>`, for an element containing directly content

- `<!ELEMENT element-name (#PCDATA|child1|otherchild1)*>`, for a mixed element
- `<!ELEMENT element-name ANY>`, for defining an element for which no further details are provided

Additionally, a DTD must specify, by using special characters, how the elements can appear (i.e. in a given order) and if they can be repeated. Specifically, the character "`,`" defines an order, "`|`" defines alternatives (*either...or*), "`()`" group elements, "`*`" indicates any number (zero or more), "`+`" means at least once (one or more) and "`?`" marks for optional (zero or one). If there is no `*`, `+` or `?`, the element must occur exactly one time.

Term	Meaning
`,`	Separates members of a sequence list and indicates sequential use of all members
`|`	Separates members of a choice list and requires use of one and only one member
`+`	Indicates a required and repeatable occurrence
`*`	Indicates an optional and repeatable occurrence
`?`	Indicates an optional occurrence

Table 1 – A table indicating the special character to formalize repetitions and order when defining elements in a DTD

For example, in case of XML documents describing books, with a title, multiple authors and different chapters, the definition of the element book will be the following:

```
<!element book (title,author+,chapter+)>
```

where we define that the element `book` can contain only other elements (not directly content) and, specifically a title (`title`), then one or more authors (`author+`) and successively one or more chapters (`chapter+`).

6.2 *Attribute-list Declaration*s

In the DTD, XML element attributes are declared with an `ATTLIST` declaration. Attribute-list declarations name the allowable set of attributes for each declared element, including the type of each attribute value, if not an explicit set of valid value(s) [22]. An attribute declaration has the following syntax:

```
<!ATTLIST elementName attributeName Type defaultValue>
```

There are the following attribute types: `CDATA` (Character set of data), `ID`, `IDREF` and `IDREFS`, `NMTOKEN` and `NMTOKENS`, `ENTITY` and `ENTITIES`, `NOTATION` and `NOTATIONS`, listings and `NOTATION`-listings. These data types are listed in Table 2

Value	Explanation
`CDATA`	The value is character data
`(eval|eval|..)`	The value must be an enumerated value
`ID`	The value is an unique id
`IDREF`	The value is the id of another element
`IDREFS`	The value is a list of other ids
`NMTOKEN`	The value is a valid XML name
`NMTOKENS`	The value is a list of valid XML names
`ENTITY`	The value is an entity
`ENTITIES`	The value is a list of entities
`NOTATION`	The value is a name of a notation
`xml:`	The value is predefined

Table 2 – A table showing the possible type of an attribute in a DTD declaration.

A default value can be used to define whether an attribute must occur (`#REQUIRED`) or not (`#IMPLIED`), whether it has a fixed value (`#FIXED`), and which value should be used as a default value ("…") in case the given attribute is left out in an XML tag.

6.3 *Valid XML documents: Including DTDs into XML*

The document type declaration, which is situated after the XML declaration, is a mechanism for naming the document type to which a document complies and for including its definition. Valid XML documents must declare the document type the follow so that editors, browsers or converters can read the DTD to understand the template structure.

Well-formed documents can also include a document type declaration and include markup declarations in its external subset but are not required to do so. The document type declaration names the document type by making reference to the root element of the document. It can make reference to an external DTD,

called the *external DTD subset*, include the DTD internally in the *internal DTD subset* or use both. Document type declarations take the general form [22]:

```
<!DOCTYPE NAME SYSTEM "file">
```

An XML document which must be compliant with respect to a DTD has the attribute `standalone` in the XML declaration set to `yes`. This means that the very first line of a document which follows a specific DTD will be the following:

```
<?xml version=”1.0” standalone=”no” […] ?>
```

A XML document is defined to be *valid* if it is a *well-formed* document and it is conforms to the rules of a given Document Type Definition (DTD). Valid XML documents offer much more to the document process than their well-formed counterparts. Document authoring, processing, storage and display are made easier because documents exist in a structured environment.

In the following, Fig. 3, we show an example of DTD and a valid XML document with respect to it.

DTD

```
<!ELEMENT message (from,to+,body) >
<!ELEMENT from #PCDATA >
<!ELEMENT to #PCDATA >
<!ELEMENT body #PCDATA >
<!ATTLIST message reply (yes|no) "no" >
```

Valid XML document

```
<?xml version="1.0" standalone=”no”
                    encoding="iso-8859-1"?>
<!DOCTYPE message SYSTEM "message.dtd">
<message reply=”yes”>
  <from>Laura</from>
  <to>John</to>
  <to>Robert</to>
  <body>this is the message body</body>
</message>
```

Fig. 3 – a DTD modeling the structure of XML documents containing messages and a valid XML document with respect to the given DTD.

7. Structuring XML Documents: XML Schemas

DTDs have been inherited by XML from its predecessor SGML, and were a good way to get XML started off quickly and give SGML people something familiar to work with. Nevertheless it soon became apparent that a more expressive solution that itself uses XML was needed.

First of all DTDs do not make use of XML syntax. Second, DTDs have no constraints on character data, meaning that if a character data is allowed, any character data is allowed. Also, for DTDs, there is not a good support for schema evolution, extension, or inheritance of declarations. For what concern elements and attributes, DTDs provide too simple attribute value models, since enumerations are clearly insufficient, they provide a too simple ID attribute mechanism and allow only default values for attributes, not for elements.

XML Schema (second edition), which became a W3C recommendation in 2004, offers a rich and flexible mechanism for defining XML vocabularies, in alternative to DTDs. The main differences between DTDs and Schemas are that the second are written in XML syntax and that are always external documents. The XML Schema specification is divided into three specifications:

- *XML Schema Part 0: Primer* [23], which intends to provide a description of the XML Schema facilities and of the language
- *XML Schema Part 1: Structures* [24] and *XML Schema Part 2: Datatypes* [25] which provide the complete normative description of the XML Schema language.

To describe all the characteristics of the XML Schema language is out of the scope of this chapter, since a book by itself would be necessary. Here, we will outline the properties of the language providing explicative examples. The reader can refer to the online specifications [26] or to the available books on this topics [6,22] to have more details.

An XML schema (called also XSD) is an XML document. It starts with the document declaration and continues by opening the root element `<schema>` and by defining the specific namespace. Within this root element all the specifications are defined. The schema ends closing the root element `</schema>`, as any well-formed XML document. Thus, in an XSD file (a simple text file with extension "`.xsd`"), the skeleton is the following:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
     [… body of the schema …]
</xsd:schema>
```

The body of the schema contains element declarations. There exist four main schema elements:

- `xsd:element` declares an element and assigns it a type
- `xsd:attribute` declares an attribute and assigns it a type
- `xsd:complexType` defines a new complex type
- `xsd:simpleType` defines a new simple type

This means that elements are declared using the element `xsd:element`, and attributes are declared using the `xsd:attribute` element. XML Schema provides a set of 19 primitive data types (e.g. boolean, string, decimal, date, time, dateTime, gYear, gDay, and NOTATION). They can be used directly in an element or attribute definition, as the examples below

```
<xsd:element name="name" type="xsd:string" />
<xsd:attribute name="age" type="xsd:integer" />
```

The two tags `xsd:complexType` and `xsd:simpleType` are used, instead, to define *new types*. Simple declarations define elements that do not have any children or attributes and can only contain text, while complex declarations describe elements that can have children and attributes as well as text.

The declarations are not themselves types, but rather an association between a name and the constraints which govern the appearance of that name in documents governed by the associated schema [22]. The following is an example of an XSD portion defining a element "`book`" of a user-defined complex type "`bookType`". Any sub-elements in the `bookType` definition is a simple element of type `string` or a number (`gYear`). The element `<xsd:sequence>` identifies a sequence of elements.

```
<xsd:element name="book" type="bookType"/>
<xsd:complexType name="bookType">
  <xsd:sequence>
    <xsd:element name="title" type="xsd:string"
      minOccurs="1" maxOccurs="1"/>
    <xsd:element name="author" type="xsd:string" />
    <xsd:element name="year" type="xsd:gYear" />
  </xsd:sequence>
</xsd:complexType>
```

XSD documents have more possibilities than DTDs for expressing cardinalities on elements belonging to a specific type. In a DTD repetitions and order can be given using special characters (Section 5) such as * or +. XSD uses

the attributes `minOccurs` and `maxOccurs` to define cardinalities. In the example above, the directive says that the element "`title`" will occur only one time within the element "`book`". Also, it is possible to define a complex type inside the element that will use it (if it will be used only for that specific element). In this case, the complex type is called *anonymous* and the syntax will be the following:

```
<xsd:element name="book">
    <xsd:complexType>
     [… complex type definition …]
    </xsd:complexType>
</xsd:element>
```

If necessary, XSD documents allow to derive new simple types from existing types, by using the `xsd:simpleType` element. It basically defines a subtype. The *name* attribute assigns a name to the new type, by which it can be referred to in a `xsd:element` type attributes. Different type of elements can be used to define the subtype. In particular:

- An `xsd:restriction` child element derives by restricting the legal values of the base type
- An `xsd:list` child element derives a type as a white space separated list of base type instances
- An `xsd:union` child element derives by combining legal values from multiple base types

In the following we present an example in which a new simple type (animal) is defined as enumeration of possible string values (dog or cat) and an element (webby) is defines of type animal.

```
<xsd:simpleType name="animal">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="dog"/>
    <xsd:enumeration value="cat"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name=”webby” type="animal" />
```

In the following we provide an example of XML Schema to define the structure of XML documents which must follow the grammar defined by the DTD in the section before

XML Schema

```
<?xml version="1.0"?>
<xsd:schema
     xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xsd:element name="message" type="messageType"/>
  <xsd:complexType name="messageType">
    <xsd:sequence>
      <xsd:element name="from" type="xsd:string"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="to" type="xsd:string"
        minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="author" type="xsd:string" />
      <xsd:element name="body" type="xsd:string"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
     <xsd:attribute
       name="reply" type="xsd:boolean" default="no"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

To complete this section, we recall that there is a multitude of tools for validating and editing schemas in XSD on the net, open source or commercial. To have an idea the reader can visit the *XML Schema Working Group* website at W3C [26].

8. Rendering XML Documents via CSS

As we have said in the previous sections, XML is born to *structure content*, not to display it. This means that, if an XML document is displayed in a browser it will be showed as text, without any formatting (actually some browsers support the user by showing the tree structure of the document, but nothing more). See, for example Fig. 4-(a). This is perfectly in line with the main goals of XML.

Anyway, there is a way in which the web representation of an XML document can be improved. Basically, a CSS style sheet can be applied similarly to what can be done (and must be done) with HTML files.

Cascading Style Sheet Language (CSS) is a language used to describe the presentation (that is, the look and formatting) of a document written in a markup language [27,28,29]. A CSS document is basically constituted by a set of rules

that must be applied to elements in the reference file. A rule consists of two parts: a *selector* and a *declaration*, with the syntax:

```
selector {property1:value; property2:value;}
```

The *selector* is the reference to the element in the file that must be rendered and the *declaration* is that part of the rule that sets forth what the effect will be. The following is an example of rule applied to an element `H1` in a HTML page.

```
H1 {color:black;}
```

CSS2 (Cascading Stylesheet Level 2) defines around 120 properties and for all of them, different values can be assigned. For a tutorial on CSS/CSS2 see, for example [28] or the last specification at [29].

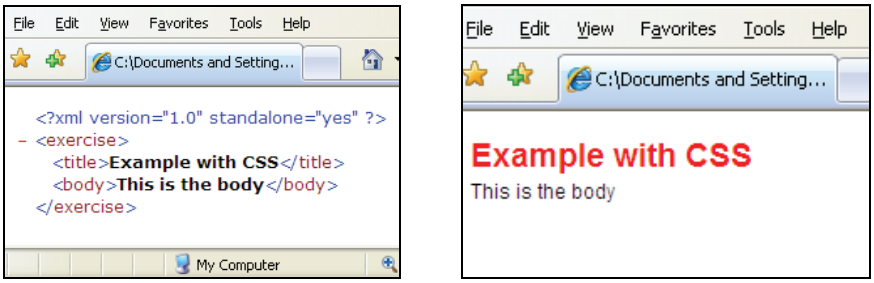
CSS/CSS2 style sheet can be easily added to HTML documents using the `link` element which create a link to the external style sheet. In XML it is possible to attach external style sheets by means of the `xml-stylesheet` processing instruction, which must be placed in the prolog of the XML document. The syntax is the following:

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

Just as with the `link` element of HTML, there can be multiple `xml-stylesheet` processing instructions, meaning that is it possible to attach multiple style sheets to an XML document. The possible attributes are *type*, *medium* and *title*, so that each stylesheet can have a local name (title), can be applied if the display medium is of a given type (print, screen..) and it has a specific type (usually text/css).

To show how attach a CSS style sheet to an XML document, we provide here a very simple example. Given the following XML code, it will be rendered in a browser as showed in Fig. 4-(a).

```
<?xml version="1.0" standalone="yes" ?>
<exercise>
  <title>Example with CSS</title>
  <body>This is the body</body>
</exercise>
```



(a) (b)

Fig. 4 – (a) an example of an XML document and how it is rendered in a browser. No specific formatting is applied. (b) the same XML document with a CSS style sheet applied.

By adding the processing instruction for including a CSS style sheet (named *style1.css*), the resulting XML code will be

```
<?xml version="1.0" standalone="yes" ?>
<?xml-stylesheet href="style1.css" type="text/css" ?>
<exercise>
 <title>Example with CSS</title>
 <body>This is the body</body>
</exercise>
```

The CSS file contains the following simple rules, one for the element `exercise` (thus it applies to the element and all its children), one for the element `title` and another for the element `body`. The results of applying style1.css to the XML document is showed in Fig. 4-(b).

```
exercise { font-family:Arial }
title    { display:block; color:red;
           font-size:14pt;
           font-weight:bold }
body     { color:black; font-size:12px }
```

Note that, even if the tags are no more visible in the browser window, the entire XML document is freely readable looking at the code of the page. Also, the way in which the information are presented follows strictly the order in which they have been modeled in the XML document (as in the case of simple HTML pages). Suppose, for example, that the initial XML document was related to a list of books for a library, it is impossible to show them in an alphabetic order, if the

books have not been inserted in that order. Additionally, if part of the content has been modeled inside attributes, there is no way to access to the attributes values and to show them in the rendered page. These are some of the limitations of CSS (CSS2) in the context of XML documents. An immediate observation is that XML is not a replacement of HTML, thus, for creating web pages, HTML (or – better- XHTML) is more than enough. XML exists for *structuring data* and means for modifying, transforming and interrogating this data are necessary.

In the following section we will show how XSL and XSLT support this type of functionalities.

9. Transforming and Rendering XML Documents: XSLT, XPath, XSL-FO

XSL is a family of recommendations for defining XML document transformation and presentation [30]. *Extensible Stylesheet Language* (languages) has the main goal to create style sheets. Basically, an XSL engine uses these style sheets to transform XML documents into other documents, and to format the output according to specific formatting templates. The XSL family consists of three main sub-languages:

- *XSL Transformations (XSLT)* [31] which is an XML-based language for transforming XML documents into other XML documents (even XHTML)
- the *XML Path Language (XPath)*, [19] which is an expression language used by XSLT to access or refer to parts of an XML document
- *XSL Formatting Objects (XSL-FO)*, [32] which is an XML vocabulary for specifying formatting semantics (that can be used instead of CSS)

In the following subsections we will briefly review the three languages, with simple examples that will support the reader in understanding how the transformation and rendering operations work on XML documents. The topic, however, is too large to be condensed in a single section of a chapter. Thus, we suggest the reader to consult books on these topics [33, 34] or the W3C online specifications [31].

Since XML documents can be represented as trees, in XSL, usually, the input document is called the *source tree*, and the output document the *result tree*.

9.1 *XSL Transformations (XSLT) and the XML Path Language (XPath)*

XSLT is a powerful language for transforming XML documents into something else that can be an HTML document, another XML document, a Portable Document Format (PDF) file, a Scalable Vector Graphics (SVG) file, a flat text file, or most anything possible [34]. The general idea is that, an XSLT stylesheet

defines the rules for transforming an XML document and the chosen XSLT processor does the work and produces the output.

XSLT relies on a technology called *XPath*. The XPath language allows XSLT identify nodes (elements, attributes, and other objects) in XML documents, as well as it provides functions for performing calculations [33].

To understand how XSLT works, we start from a XML document and we apply a XSLT template to transform the content of this document in a HTML page. The input XML document is the following:

```
<?xml version="1.0" standalone="yes" ?>
<TitleBook>This title will become H1</TitleBook>
```

It is a very simple document, with only the root element `TitleBook` which contains directly the content, with no other sub-elements. The objective of the XSLT transformation we are going to produce, is to take the content in the element `TitleBook` and to put it inside an `H1` tag of a HTML page.

Recall that an XSLT transformation file is, first of all, an XML document, thus it follows the same syntax of any other XML. Also, in order to "use" the XSLT language we have to define the appropriate namespace. The skeleton of transformation file will be:

```
<?xml version="1.0" standalone="yes" ?>
<xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<output method="html"/>
   [… other directives …]
</xsl:stylesheet>
```

The first line is the XML declaration, the second defines the root element `<xsl:stylesheet>` and the XSLT namespace (prefix `xsl:`) with the official W3C URI `http://www.w3.org/1999/XSL/Transform.` The third line, instead, provides a directive on the output method, namely HTML. As any well-formed XML, the style sheet ends with the root element closing tag, in this case, `</xsl:stylesheet>`.

The other necessary directives are few and simple. First of all we need to intercept the root element and we need to apply a stylesheet template on it, taking the content associate and rewriting it as content of the HTML tag H1.

For doing this, we use the following code:

```
<xsl:template match = "/" >
    <html><body>
      <h1><xsl:value-of select = "TitleBook" /></h1>
     </body></html>
</xsl:template>
```

Basically, we define a *template* and we apply it to the XML portion which is described in the attribute `match`. In this case XPath language is used to intercept the desired element. In the example, the expression "`/`" identifies the root element (similarly to what we can do in a file system). There are different possible XPath expressions the can be used which allow to fetch every element, entity or attribute in the document. Table 3 provides a selection of XPath expressions [35].

Name	**Return value**
Node-Set-oriented	
`last()`	Number (of elements)
`count(node-set)`	Number of nodes in node-set
`name(node-set?)`	First node name in node-set
Sign oriented	
`concat(string,string*)`	Sequence of arguments
`starts-with(string,string)`	True if first string begins with second one
`contains(string,string)`	True if first string includes second one
Boolean	
`not()`	True if argument is not true
`true()/false()`	True/not true

Table 3 – a selection of XPath expressions, divided into node oriented, sign oriented and Boolean

Once the root element (node in the source tree) has been selected, we "extract" the content using another XSLT directive `<xsl:value-of…>`. It has an attribute `select` which contains another XPath expression. In the case of the example, we extract the content inside the element `TitleBook`. Finally, by putting the necessary HTML tags and the `h1` tag "around" the extracted content, we create the web page.

Taking a XSLT processor (even the modern browsers support this feature) and giving in input both the XML and the XSLT documents, it interprets the

XSLT directive and creates the new HTML page. Actually, XML, XSLT, and XPath are correctly supported by the following browsers: Mozilla Firefox 3, Internet Explorer 6+, Google Chrome, Opera 9 and Apple Safari 3+.

The operational schema of an XSLT transformation is the one presented in Fig. 5.

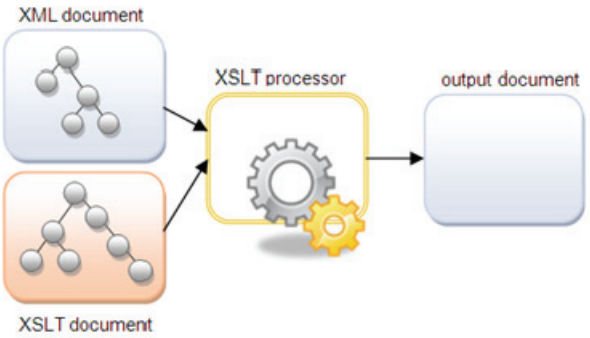


Fig. 5 – one or more XML documents with one or more XSLT transformations are passed to the XSLT processor which builds the output document.

The element `<template>` is the main element of a XSLT document. The number of elements (directives) it can contain is very high. We provide here some examples to let the user understand the power of the language.

A template can be iteratively applied to elements in the XML document using different tags, as, for example `<xsl:for-each …>`. This is the case of a loop for each element satisfying the XPath expression inside the attribute `select` of the `<xsl:for-each …>` tag. For example the following code applies a template, selecting the content of each `paragraph` inside a `chapter` in the input XML document:

```
<xsl:template match="chapter">
  <xsl:for-each select="paragraph">
     <xsl:value-of select=".">
  </xsl:for-each>
</xsl:template>
```

Table 4 presents a set of elements for XSLT to provide an idea of the main functionalities

Element syntax	Meaning and Notes

`<xsl:if test="…">`
`…`
`</xsl:if>`
	Yes or No conditions
	The condition is inside attribute `test`

`<xsl:variable`
`name="type"`
`select="@type"/>`
	Allows a variable to be declared
	`name` is the variable name. It can be referred to later with `$name` `select` is the value of the variable

`<xsl:when test="…">`
`…`
`</xsl:when>`
	Yes or No conditions
	`test` specifies criteria for entering the if

`<xsl:choose>`
`…`
`</xsl:choose>`
	Multiple choices
	No attributes

`<xsl:for-each`
`select="…">`
`…`
`</xsl:for-each>`
	Creates a loop which repeats for every match. `select` designates the match criteria

`<xsl:apply-templates/>`	Specifies that other matches may exist within that node; if this is not specified any matches will be ignored If `select` is specified, only the templates that specify a `match` that fits the selected node or attribute type will be applied. If `mode` is specified, only the templates that have the same `mode` and have an appropriate `match` will be applied

`<xsl:text>`
`a text`
`</xsl:text>`
	Outputs the tag content

`<xsl:value-of` `select="$s"/>`	Outputs a variable `Select` specifies the variable

Table 4 – few examples of elements in XSLT for capturing, choosing content in XML documents, as well as defining templates of transformations on the basis of specific conditions.

9.2 *XSL Formatting Objects* XSL-FO

XSL Formatting Objects (or Flow Objects) [32], or XSL-FO, is a XML-based markup language which describes the formatting of XML data for output to

screen, paper or other media. Thanks to XSL-FO it is possible to produce, for example, documents in PDF format, RTF or even PS, starting from an input XML document. XSL-FO covers the basic presentation requirements for a wide range of display devices, including reflow or repagination for palmtop devices, and for the accessibility requirements that are now mandated by governments.

XSL-FO is used inside XSLT transformations. As described in the above subsection, an XSLT transformation take an XML document (source tree) and gives the directive to produce a result tree (the work is done by the processor, which interprets the directives). Once transformed, the operation of *formatting* – done by the XSL processor - interprets the result tree looking at the formatting objects contained into the directives specialized with the XSL-FO language.

The XSL-FO language was designed for paged media, thus the concept of *page* is an important part of its structure, and the formatting objects it provides give significant expressive power in dealing with how information is displayed on a page.

So, basically, XSL-FO documents are XML files with output information, usually stored in files with `.fo` or `.fob` extension. They contain two required sections: (*i*) the first section details a list of named page layouts; (*ii*) the second section is a list of document data, with markup, that uses the various page layouts to determine how the content fills the various pages. The skeleton of an XSL-FO document is the following:

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="A4">
      <!-- Page template goes here -->
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="A4">
     <!-- Page content goes here -->
  </fo:page-sequence>
</fo:root>
```

As for the other XML-based languages described before, a XSL-FO document starts with a root element `<fo:root>` containing the appropriate namespace declaration, namely "`http://www.w3.org/1999/XSL/Format`". Two main elements are successively declared:

- `fo:layout-master-set` which contains the collection of definitions of page geometries and page selection patterns
- `fo:page-sequence` which contains the definition of information for a sequence of pages with common static information

The interpretation of the two main elements above it the following: when the formatter reads the XSL-FO document, it creates a page based on the first template in the `fo:layout-master-set`. Then it fills it with content from the `fo:page-sequence`. When it's filled the first page, it instantiates a second page based on a template, and fills it with content. The process continues until the formatter runs out of content [22].

9.3 *Page Formatting*

The page templates are called *page masters*. Each defines a general layout for a page including its margins, the sizes of the header, footer, and body area of the page, and so forth. XSL-FO 1.0 defines exactly one kind of page master, the `fo:simple-page-master`, which represents a rectangular page. The `fo:layout-master-set` contains one or more `fo:simple-page-master` elements that define master pages.

For example, we present in the following portion of XSL-FO code a `fo:layout-master-set` containing one `fo:simple-page-master`. It contains a single region, the body, into which all content will be placed.

```
<fo:layout-master-set>
  <fo:simple-page-master  master-name="…"
     page-height=".."   page-width=".." […]>
    <fo:region-body/>
  </fo:simple-page-master>
</fo:layout-master-set>
```

9.4 *Page sequence management*

In addition to a `fo:layout-master-set`, as we said before, each formatting object document contains one or more `fo:page-sequence` elements. In this case, the XSL-FO specifies the sequence of pages, where each page has an associated page master that defines how the page will look. Each page sequence contains three child elements in this order:

(i) An optional `fo:title` element containing inline content that can be used as the title of the document.
(ii) Zero or more `fo:static-content` elements containing the for every page
(iii) One `fo:flow` element containing data to be placed on each page in turn (in case of pagination)

The following is an example of code for defining the pages sequence:

```
<fo:page-sequence master-reference="chaps">
  <fo:static-content flow-name="…">
    <fo:block text-align="outside" …>
      Chapter
      <fo:retrieve-marker
                retrieve-class-name="chapNum"/>
      <fo:leader leader-pattern="space" />
      <fo:retrieve-marker retrieve-class-name="chap"/>
      <fo:leader leader-pattern="space" />
      Page
      <fo:page-number font-style="normal" />
      of
      <fo:page-number-citation ref-id='end'/>
    </fo:block>
  </fo:static-content>
  <fo:flow flow-name="…">
   <fo:block>
     <!-- Output goes here -->
   </fo:block>
  </fo:flow>
</fo:page-sequence>
```

In the example, the sequence of pages is defined for chapters in a book and the portion of document gives directives for the rendering of the chapter numbers, the page number and other information.

9.5 *Formatting Objects*

The vocabulary of formatting objects supported by XSL-FO represents the set of typographic abstractions available. Some of them are very similar to properties that can be found in CSS. For example, it is possible to enrich text with

character-level formatting. There exist several properties control font styles — family, size, color, weight, etc. The following is an example:

```
<fo:block font-family="Times" font-size="14pt">
  This text will be Times of size 14pt!
</fo:block>
```

Other formatting objects are specialized to described the output on different media (pagination, borders and so on). Each formatting object class represents a particular kind of formatting behavior. For example, the `block` formatting object class represents the breaking of the content of a paragraph into lines [32,37]. The following is an example:

```
<fo:block line-height="1.0" text-align="justify">
  Example of a justified formatted block.
  The space between lines is 1.0.
</fo:block>
```

Another example is the `list-item` formatting object which is a box containing two other formatting objects, namely `list-item-label` and `list-item-body`. The first one (`list-item-label`) is basically a box that contains a bullet, a number, or another indicator placed in front of a list item, the second one (`list-item-body`) is a box that contains the text of the list item.

Tables, lists, side floats, and a variety of other features are available. These features are comparable to CSS's layout features, though some of those features are expected to be built by the XSLT even before the application of XSL-FO, if necessary.

To conclude this section, we outline that there is a multitude of processors which are able to interpret the XSL family of technologies. The reader can refer to the official XSL web page at W3C for a complete list [30].

10. Conclusions and Outlook

In this chapter we have provided an introduction to XML, presenting its main goals and trying to focus on the great impact it had in the development of the modern Web Technologies. We outlined how XML is a meta-language for defining new languages potentially on every application domain.

We provided notions on the syntax and the ways in which organizations, companies and institutions can structure the content of XML documents by using

DTDs and XML Schemas. We reviewed also how XML documents can be rendered, using CSS style sheets and how they can be transformed and rendered using XSL/XSLT, which are a powerful XML-based languages for creating directives to deal with XML documents.

It can be easily understood, by simply searching on the web the "XML" word, which is the incredible impact of XML in the scientific and industrial scenarios. It has been proved to be a powerful means to allow interoperability and to improve communications among business entities, which has emerged to be a real necessity thanks also the evolution of the Web. Looking at the W3C home page, it is clear how many different technologies have been developed upon pr around XML. Several working groups and activities have been defined and are active in many different topics related to XML.

In the context of this book, the Semantic Web Activity is maybe the most interesting [38]. The Semantic Web is a web of data, as the reader will discover in the other chapters of this book. This activity includes different recommendations, most of them designed on XML, as for example:

- Resource Description Framework (RDF) [12] is an XML text format that supports resource description and metadata applications.
- GRDDL [40] is a mechanism for gleaning resource descriptions from dialects of languages. It introduces markup based on existing standards for declaring that an XML document includes data compatible with the Resource Description Framework (RDF) and for linking to algorithms (typically represented in XSLT), for extracting this data from the document.
- The Web Ontology Language OWL [39] is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language.
- SPARQL query language for RDF, which can be used to express queries across diverse data sources [41].

However, for sake of completeness, we would like to remember that there are many other standards and on-going works of XML-based languages, as for example:

- XHTML: This is the XML-version of HTML 4.0 [9].
- Chemical Markup Language (CML): CML is used for molecular information management. [10].
- Simple Object Access Protocol (SOAP): A protocol that is object based and used for information exchange in a decentralized and distributed environment.[11]

- Synchronized Multimedia Integration Language (SMIL, pronounced "smile"). SMIL 3.0 defines an XML-based language that allows authors to write interactive multimedia presentations.
- Scalable Vector Graphics (SVG), is a language for describing two-dimensional graphics and graphical applications in XML
- XML Query (XQuery), is a standardized language for combining documents, databases, Web pages and almost anything else.
- WSDL: Web Service Description Language. An XML format for describing XML web services, including the type definitions, messages and actions used by that service. The WSDL document should tell applications all they need to know to invoke a particular web service

Acknowledgments

This work was supported by the University of Genova. I would like to thank all the authors of existing books and online tutorials on XML who, being also on the web, allow to spread the knowledge on this powerful technology.

11. References

1. Extensible Markup Language (XML) 1.0 Fifth Edition (2008). W3C Recommendation, Eds. T. Bray, J.Paoli, C. M. Sperberg-McQueen, E.Maler, F.Yergeau, www.w3.org/TR/REC-xml/
2. XML, Wikipedia, the free encyclopedia (last access 2009), http://en.wikipedia.org/wiki/XML
3. St. Laurent, Simon. (1999). XML: A Primer. Foster City: M&T Books.
4. Bryan, Martin. (1998). "An Introduction to the Extensible Markup Language (XML)." Bulletin of the American Society for Information Science 25, 1. 11-14.
5. Introduction to XML, W3Schools (last access 2009), www.w3schools.com/XML/
6. Møller Anders and Schwartzbach Michael I, An Introduction to XML and Web Technologies, Addison-Wesley, ISBN: 0321269667 January 2006
7. A.Attipoe and P.Vijghen (1999). "XML/SGML: On the Web and Behind the Web." InterChange: Newsletter of the International SGML/XML Users' Group Volume 5, Issue 3, pages 25-29
8. Jon Bosak, (2003), The Birth of XML: A Personal Recollection, http://java.sun.com/xml/
9. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) A Reformulation of HTML 4 in XML 1.0 (2002) W3C Recommendation www.w3.org/TR/html/
10. P.M. Rust and H.S. Rzepa (1999), Chemical Markup, XML, and the World Wide Web. Basic Principles, J. Chem. Inf. Comput. Sci., 39, 928-942
11. SOAP Version 1.2 Part 0: Primer (Second Edition) (2007), W3C Recommendation www.w3.org/TR/soap/
12. Resource Description Framework (RDF), (2004) W3C Recommendation, www.w3.org/RDF
13. XML namespace, Wikipedia, the free encyclopedia (last access 2009) http://en.wikipedia.org/wiki/XML_namespace

14. Namespaces in XML 1.0 (Second Edition) (2006), W3C Recommendation, www.w3.org/TR/xml-names/
15. R. Srivastava (last access 2009) XML Schema: Understanding Namespaces, Oracle www.oracle.com/technology/pub/articles/srivastava_namespaces.html
16. Uniform Resource Identifier, (last access 2009) Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Uniform_Resource_Identifier
17. XML Linking Language (XLink) Version 1.0, (2001), W3C Recommendation, www.w3.org/TR/xlink/
18. FOAF Vocabulary Specification 0.91, (2007), Namespace Document OpenID Edition, http://xmlns.com/foaf/spec/
19. XML Path Language (XPath) Version 1.0 (1999), W3C Recommendation www.w3.org/TR/xpath
20. E. Maler, (last access 2009) Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1, www.w3.org/XML/1998/06/xmlspec-report-v21.htm
21. XML Core Working Group Public Page (last access 2009), www.w3.org/XML/Core/
22. E.R. Harold XML Bible, 2nd edition (2001), John Wiley & Sons, Inc. New York, NY, USA
23. XML Schema Part 0: Primer Second Edition, (2004) W3C Recommendation, Eds. David C. Fallside, Priscilla Walmsley - www.w3.org/TR/xmlschema-0/
24. XML Schema Part 1: Structures Second Edition, (2004) W3C Recommendation, Eds. H.S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, www.w3.org/TR/xmlschema-1/
25. XML Schema Part 2: Datatypes Second Edition, (2004) W3C Recommendation, Eds. P.V. Biron, K.Permanente, A.Malhotra, www.w3.org/TR/xmlschema-2/
26. The XML Schema Working Group (2001) - www.w3.org/XML/Schema
27. Cascading Style Sheets Home Page (2009) - www.w3.org/Style/CSS/
28. H.W Lie and B. Bos (1999). Cascading Style Sheets, designing for the Web, 2nd edition, Addison Wesley, ISBN 0-201-59625-3
29. Cascading Style Sheets Level 2 (CSS 2.1) Specification (2009), W3C Candidate Recommendation, Eds. B. Bos, T. Çelik, I. Hickson , H.W. Lie www.w3.org/TR/REC-CSS2
30. The Extensible Stylesheet Language Family (XSL), http://www.w3.org/Style/XSL/
31. XSL Transformations (XSLT) Version 2.0 (2007) W3C Recommendation www.w3.org/TR/xslt20/
32. Extensible Stylesheet Language (XSL) Version 1.1 (2006), W3C Recommendation www.w3.org/TR/xsl/
33. M. Fitzgerald (2003), Learning XSLT, O'Reilly Press. ISBN 10: 0-596-00327-7
34. D. Tidwell (2008), XSLT, Second Edition, Mastering XML Transformations O'Reilly Press. ISBN 10: 0-596-52721-7
35. Michael Kay; XSLT Programmer's Reference; Birmingham (Wrox Press) 2000
36. G.K. Holman, (2002), What Is XSL-FO, online article www.xml.com/pub/a/2002/03/20/xsl-fo.html
37. How to Develop Stylesheet for XML to XSL-FO Transformation (2007), Antenna House, Inc. www.antennahouse.com/XSLsample/howtoRC/Howtodevelop-en-2a.pdf
38. The Semantic Web Activity, W3C, www.w3.org/2001/sw/
39. The Web Ontology Language OWL, (2004), W3C Recommendation, www.w3.org/TR/owl-ref/

40. Gleaning Resource Descriptions from Dialects of Languages (GRDDL), (2007) W3C Recommendation www.w3.org/TR/grddl/
41. SPARQL Query Language for RDF, (2008), W3C Recommendation www.w3.org/TR/rdf-sparql-query/