http://webdam.inria.fr/

# Web Data Management

## Data model

Serge Abiteboul
INRIA Saclay & ENS Cachan

Ioana Manolescu
INRIA Saclay & Paris-Sud University

Philippe Rigaux
CNAM Paris & INRIA Saclay

Marie-Christine Rousset
Grenoble University

Pierre Senellart
Télécom ParisTech



http://webdam.inria.fr/Jorge/

# Contents



The Web is a media of primary interest for companies who change their organization to place it at the core of their operation. It is an easy but boring task to list areas where the Web can be usefully leveraged to improve the functionalities of existing systems. One can cite in particular B2B and B2C (business to business or business to customer) applications, G2B and G2C (government to business or government to customer) applications or digital libraries. Such applications typically require some form of typing to represent data because they consist of programs that deal with HTML text with difficulties. Exchange and exploitation of business information call as well for a more powerful Web data management approach.

This motivated the introduction of a semistructured data model, namely XML, that is well suited both for humans and machines. XML describes *content* and promotes machine-to-machine communication and data exchange. The design of XML relies on two major goals. First it is designed as a generic data format, apt to be specialized for a wide range of data usages. In the XML world for instance, XHTML is seen as a specialized XML dialect for data presentation by Web browsers. Second XML "documents" are meant to be easily and safely transmitted on the Internet, by including in particular a self-description of their encoding and content.

XML is the language of choice for a generic, scalable, and expressive management of Web data. In this perspective, the visual information between humans enabled by HTML is just a very specific instance of a more general data exchange mechanism. HTML also permits a limited integrated presentation of various Web sources (see any Web portal for instance). Leveraging these capabilities to software-based information processing and distributed management of data just turns out to be a natural extension of the initial Web vision.

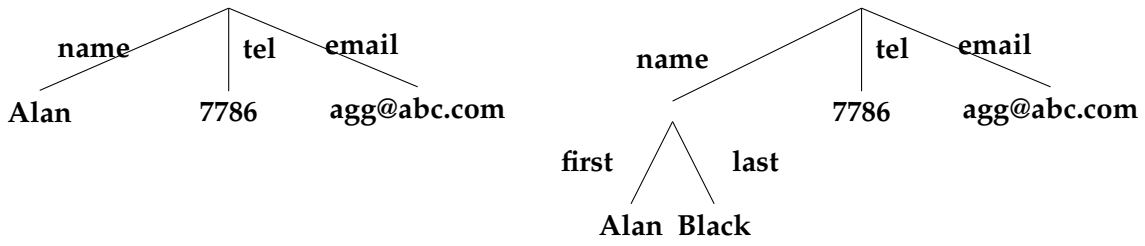



Figure 1: Tree representation, with labels on edges

The chapter first sketches the main traits of semistructured data models. Then we delve into XML and the world of Web standards around XML.

# 1 Semistructured data

A semistructured data model is based on an organization of data in labeled trees (possibly graphs) and on query languages for accessing and updating data. The labels capture the structural information. Since these models are considered in the context of data exchange, they typically propose some form of data serialization (i.e., a standard representation of data in files). Indeed, the most successful such model, namely XML, is often confused with its serialization syntax.

Semistructured data models are meant to represent information from very structured to very unstructured kinds, and, in particular, irregular data. In a structured data model such as the relational model, one distinguishes between the type of the data (*schema* in relational terminology) and the data itself (*instance* in relational terminology). In semistructured data models, this distinction is blurred. One sometimes speaks of schema-less data although it is more appropriate to speak of self-describing data. Semistructured data may possibly be typed. For instance, tree automata have been considered for typing XML (see Chapter **??**). However, semistructured data applications typically use very flexible and tolerant typing; sometimes no typing at all.

We next present informally a standard semistructured data model. We start with an idea familiar to Lisp programmers of association lists, which are nothing more than label-value pairs and are used to represent record-like or tuple-like structures:

```
{name: "Alan", tel: 2157786,  email: "agb@abc.com"}
```

This is simply a set of pairs such as (`name`, `"Alan"`) consisting of a label and a value. The values may themselves be other structures as in

```
{name: {first: "Alan",  last: "Black"},
 tel: 2157786,
 email: "agb@abc.com"}
```

We may represent this data graphically as a tree. See, for instance, Figures 1 and 2. In Figure 1, the label structure is captured by tree edges, whereas data values reside at leaves. In Figure 2, the second, all information resides in the vertices.

Such representations suggest departing from the usual assumption made about tuples or association lists that the labels are unique, and we allow duplicate labels as in

```
{name: "Alan", tel: 2157786, tel: 2498762 }
```

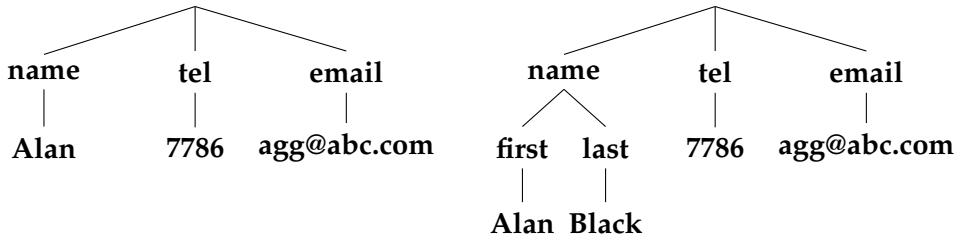



Figure 2: Tree representation, with labels as nodes

The syntax makes it easy to describe sets of tuples as in

```
{ person:  {name: "Alan", phone: 3127786, email: "alan@abc.com"},
  person:  {name: "Sara", phone: 2136877, email: "sara@xyz.edu"},
  person:  {name: "Fred", phone: 7786312, email: "fd@ac.uk"} }
```

Furthermore, one of the main strengths of semistructured data is its ability to accommodate variations in structure (e.g., all the `Person` tuples do not need to have the same type). The variations typically consist of missing data, duplicated fields, or minor changes in representation, as in the following example:

```
{person: {name: "Alan", phone: 3127786, email: "agg@abc.com"},
 person: &314
         {name: {first: "Sara", last: "Green"              },
          phone: 2136877,
          email: "sara@math.xyz.edu",
          spouse: &443                                     },
 person: &443
         {name: "Fred", Phone: 7786312, Height: 183,
          spouse: &314                                     }}
```

Observe how identifiers (here `&443` and `&314`) and references are used to represent graph data. It should be obvious by now that a wide range of data structures, including those of the relational and object database models, can be described with this format.

As already mentioned, in semistructured data, we make the conscious decision of possibly not caring about the type the data might have and serialize it by annotating each data item explicitly with its description (such as `name`, `phone`, etc.). Such data is called *self-describing*. The term "serialization" means converting the data into a byte stream that can be easily transmitted and reconstructed at the receiver. Of course, self-describing data wastes space, since we need to repeat these descriptions with each data item, but we gain interoperability, which is crucial in the Web context.

There have been different proposals for semistructured data models. They differ in choices such as: labels on nodes *vs.* on edges, trees *vs.* graphs, ordered trees *vs.* unordered trees. Most importantly, they differ in the languages they offer. Two quite popular models (at the time of writing) are XML, a de facto standard for exchanging data of any kind, and JSON ("Javascript Object Notation"), an object serialization format mostly used in programming environments. We next focus on XML, an introduction to JSON being given in Chapter **??**.

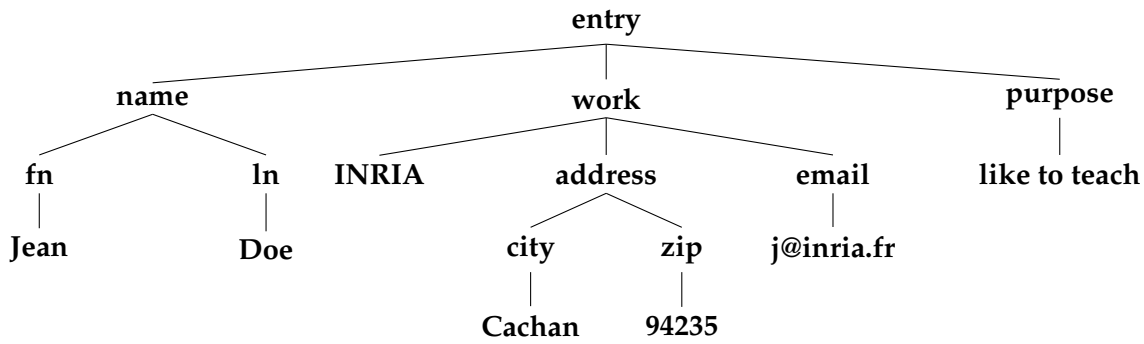



Figure 3: An XML document

# 2 XML

XML, the Extensible Markup Language, is a semistructured data model that has been proposed as the standard for data exchange on the Web. It is a simplified version of SGML (ISO 8879). XML meets the requirements of a flexible, generic, and platform-independent language, as presented earlier. Any XML document can be serialized with a normalized encoding, for storage or transmission on the Internet.

**Remark 2.1** It is well-established to use the term "XML document" to denote a hierarchically structured content represented with XML conventions. Although we adopt this standard terminology, please keep in mind that by "document" we mean both the content and its structure, but not their specific representation which may take many forms. Also note that "document" is reminiscent of the SGML application area, which mostly focuses on representating technical documentation. An XML document is not restricted to textual, human-readable data, and can actually represent any kind of information, including images, of references to other data sources.

XML is a standard for representing data but it is also a family of standards (some in progress) for the management of information at a world scale: XLink, XPointer, XML Schema, DOM, SAX, XPath, XSL, XQuery, SOAP, WSDL, and so forth.

## 2.1 XML documents

An XML document is a labeled, unranked, ordered tree:

**Labeled** means that some annotation, the label, is attached to each node.

**Unranked** means that there is no a priori bound on the number of children of a node.

**Ordered** means that there is an order between the children of each node.

The document of Figure 3 can be serialized as follows:

```
<entry><name><fn>Jean</fn><ln>Doe</ln></name>INRIA<adress><city>
Cachan</city><zip>94235</zip></adress><email>j@inria.fr</email>
</job><purpose>like to teach</purpose></entry>
```

or with some beautification as

```
<entry>
   <name>
      <fn>Jean</fn>
      <ln>Doe</ln> </name>
   <work>
      INRIA
      <adress>
         <city>Cachan</city>
         <zip>94235</zip> </adress>
      <email>j@inria.fr</email> </work>
   <purpose>like to teach</purpose>
</entry>
```

In this serialization, the data corresponding to the subtree with root labeled (e.g., *work*), is represented by a subword delimited by an opening tag `<work>` and a closing tag `</work>`. One should never forget that this is just a serialization. The conceptual (and mathematical) view of an XML document is that it is a *labeled, unranked, ordered tree*.

XML specifies a "syntax" and no a priori semantics. So, it specifies the content of a document but not its behavior or how it should be processed. The labels have no predefined meaning unlike in HTLM, where, for example, the label `href` indicates a reference and `img` an image. Clearly, the labels will be assigned meaning by applications.

In HTML, one uses a predefined (finite) set of labels that are meant primarily for document presentation. For instance, consider the following HTML document:

```
<h1> Bibliography </h1>
  <p> <i> Foundations of Databases </i>
          Abiteboul, Hull, Vianu
         <br/> Addison Wesley, 1995 </p>
  <p> <i> Data on the Web </i>
          Abiteboul, Buneman, Suciu
         <br/> Morgan Kaufmann, 1999 </p>
```

where `<h1>` indicates a title, `<p>` a paragraph, `<i>` italics and `<br/>` a line break (`<br/>` is both an opening and a closing tag, gathered in a concise syntax equivalent to `<br></br>`). Observe that this is in fact an XML document; more precisely this text is in a particular XML dialect, called XHTML. (HTML is more tolerant and would, for instance, allow omitting the `</p>` closing tags.)

The presentation of that HTML document by a classical browser can be found in Figure 4. The layout of the document depends closely on the interpretation of these labels by the browser. One would like to use different layouts depending on the usage (e.g., for a mobile phone or for blind people). A solution for this is to separate the content of the document and its layout so that one can generate different layouts based on the actual software that is used to present the document. Indeed, early on, Tim Berners-Lee (the creator of HTML) advocated the need for a language that would go beyond HTML and distinguish between content and presentation.

The same bibliographical information is found, for instance, in the following XML document:

```
<bibliography>
  <book>
    <title> Foundations of Databases </title>
    <author> Abiteboul </author> <author> Hull </author>
    <author> Vianu </author>
    <publisher> Addison Wesley </publisher>
    <year> 1995 </year> </book>
  <book>...</book>
</bibliography>
```

Observe that it does not include any indication of presentation. There is a need for a stylesheet (providing transformation rules) to obtain a decent presentation such as that of the HTML document. On the other hand, with different stylesheets, one can obtain documents for several media (e.g., also for PDF). Also, the tags produce some semantic information that can be used by applications, (e.g., *Addison Wesley is the publisher of the book*). Such tag information turns out to be useful for instance to support more precise search than that provided by Web browser or to integrate information from various sources.

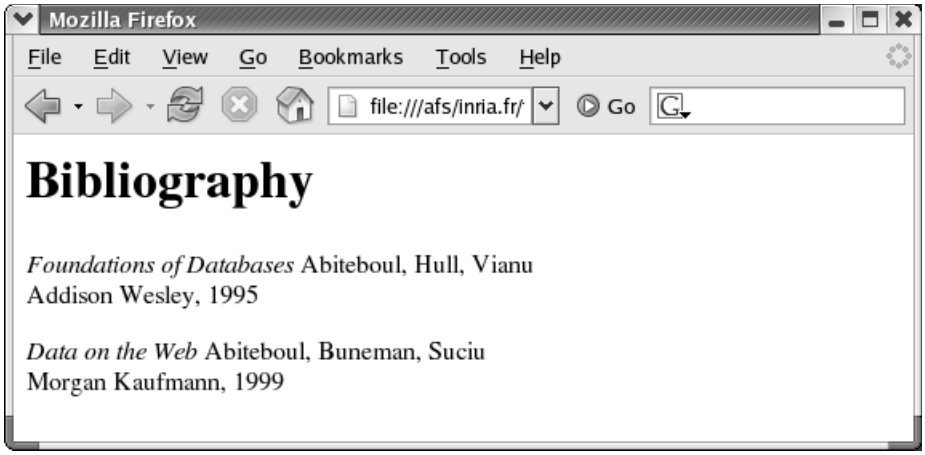



Figure 4: HTML presentation

The separation between content and presentation already exists in a precursor of XML, namely SGML. In SGML, the labels are used to describe the structure of the content and not the presentation. SGML was already quite popular at the time XML was proposed, in particular for technical documentation (e.g., Airbus documentation). However, SGML is unnecessarily complicated, in particular with features found in complex document models (such as footnote). XML is much simpler. Like SGML, it is a metalanguage in that it is always possible to introduce new tags.

## 2.2 Serialized and tree-based forms

An XML document must always be interpreted as a tree. However the tree may be represented in several forms, all equivalent (i.e., there exists a mapping from one form to another) but quite different with respect to their use. All the representations belong to one of the following category:

- *serialized forms*, which are textual, linear representations of the tree that conform to a (sometimes complicated) syntax;
- *tree-based forms*, which implement, in a specific context (e.g., object-oriented models), the abstract tree representation.

Both categories cover many possible variants. The syntax of the serialized form makes it possible to organize "physically" an XML document in many ways, whereas tree-based forms depend on the specific paradigm and/or technique used for the manipulation of the document. A basic pre-requisite of XML data manipulation is to know the main features of the serialized and tree-based representation, and to understand the mapping that transforms one form to another.

Figure 5 shows the steps typically involved in the processing of an XML document (say, for instance, editing the document). Initially, the document is most often obtained in serialized form, either because it is stored in a file or a database, or because it comes from another application. The *parser* transforms the serialized representation to a tree-based representation, which is conveniently used to process the document content. Once the application task is finished, another, complementary module, the *serializer*, transforms the tree-based representation of the possibly modified document into one of its possible serialized forms.

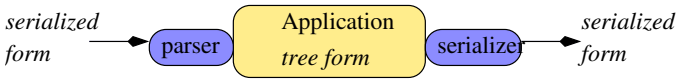



Figure 5: Typical processing of XML data

Stricly speaking, the *syntax* of XML relates to its serialized representation. The syntax can be normalized because a serialized document is meant for data exchange in an heterogeneous environment, and must, therefore, be completely independent from a specific context. The tree-based representation is more strongly associated with the application that processes the document, and in particular to the programming language.

We provide a presentation of the XML syntax that covers the main aspects of the serialized representation of an XML document and show their couterpart in terms of a tree-based representation. The serialized syntax is defined by the World Wide Web Consortium (W3C) and can be found in the XML 1.0 recommendation. Since the full syntax of XML is rather complex and contains many technical detail that do not bring much to the understanding of the model, the reader is referred to this recommendation for a complete coverage of the standard (see the last section).

For the tree-based representation, we adopt the DOM (Document Object Model), also standardized by the W3C, which defines a common framework for the manipulation of

documents in an object-oriented context. Actually we only consider the aspects of the model that suffice to cover the tree representation of XML documents and illustrate the transformation from the serialized form to the tree form, back and forth. The DOM model is also convenient to explain the semantics of the XPath, XSLT and XQuery languages, presented in the next chapters.

## 2.3 XML syntax

Four examples of XML documents (separated by blank lines) are:

```
<document/>
```

Document 1

```
<document>
  Hello World!
</document>
```

Document 2

```
<document>
  <salutation>
    Hello World!
  </salutation>
</document>
```

Document 3

```
<?xml version="1.0"
  encoding="utf-8" ?>
<document>
  <salutation color="blue">
   Hello World!
  </salutation>
</document>
```

Document 4

In the last one, the first line starting with `<?xml` is the *prologue*. It provides indications such as the version of XML that is used, the particular coding, possibly indications of external resources that are needed to construct the document.

### Elements and text

The basic components of an XML document are *element* and *text*. The text (e.g., *Hello World!*), is in UNICODE. Thus texts in virtually all alphabets, including, for example, Latin, Hebrew, or Chinese, can be represented. An element is of the form

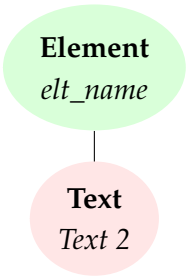
```
<name attr='value' ...> content </name>
```

where `<name>` is the *opening tag* and `</name>` the *closing tag*.

The content of an element is a list of text or (sub) elements (and gadgets such as comments). A simple and very common pattern is a combination of an element and a textual content. In the serialized form, the combination appears as

```
<elt_name>
  Textual  content
</elt_name>
```

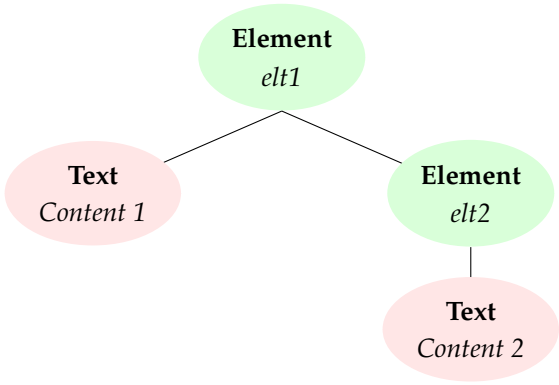
The equivalent tree-based representation consists of *two* nodes, one that corresponds to the structure marked by the opening and closing tags, and the second, child of the first, which corresponds to the textual content. In the DOM, these nodes are typed, and the tree is represented as follows:




The **Element** nodes are the internal nodes of a DOM representation. They represent the hierarchical structure of the document. An **Element** node has a *name*, which is the label of the corresponding tag in the serialized form. The second type of node illustrated by this example is a **Text** node. **Text** nodes do not have a name, but a *value* which is a non structured character string.

The nesting of tags in the serialized representation is represented by a parent-child relationship in the tree-based representation. The following is a slight modification of the previous examples which shows a nested serialized representation (on the left) and its equivalent tree-based representation (on the right) as a hierarchical organization with two **Element** nodes and two **Text** nodes.

```
<elt1>
  Content 1
  <elt2>
    Content 2
  </elt2>
</elt1>
```




### Attributes

The opening tag may include a list of (name,value) pairs called *attributes* as in:

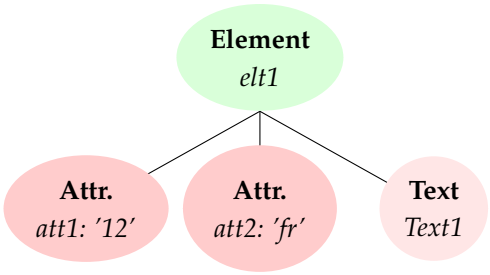
```
<report language='fr' date='08/07/07'>
```

Two pairs of attributes for the same element are not allowed to have the same attribute name.

Major differences between the content and the attributes of a given element are that (i) the content is ordered whereas the attributes are not and (ii) the content may contain some complex subtrees whereas the attribute value is atomic.

Attributes appear either as pairs of name/value in opening tag in the serialized form, or as special child nodes of the **Element** node in the tree-based (DOM) representation. The following example shows an XML fragment in serialized form and its counterpart in tree-based form. Note that **Attr** nodes have both a name *and* a value.

```
<elt att1='12' att2='fr'>
  Text1
</elt>
```



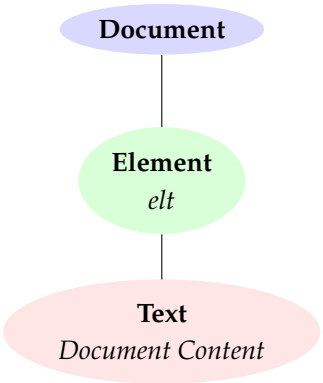

Attribute can store content, just as **Text** nodes. In the previous example, the textual content could just be represented as an attribute of the `elt` element, and conversely attributes could be represented as child elements with a textual content. This gives rise to some freedom to organize the content of an XML document and adds some complexity to the tree-based representation.

### Well-formed XML document

An XML document must correctly represent a tree. There exist in a document one and only one element that contains all the others (called the *root element*). An element that is not the root is totally included in its parent. More generally, the tags must close in the opposite order they have been opened. One says of such a document that it is *well-formed*. For instance, `<a> <b> </b> </a>` is well-formed and `<a> <b> </a> </b>` is not.

The serialized form often (but not always) begins with the prologue; independently of the existence or not of a prologue, the tree-based representation of an XML document has for root a **Document** node. This node has a *unique* **Element** child, which is the *root element* of the document. The following examples illustrates the situation.

```
<?xml version="1.0"
    encoding="utf-8" ?>
<elt>
  Document content.
</elt>
```




There may be other syntactic objects after the prologue (for instance, processing instructions), which become children of the **Document** node in the tree representation.

The **Document** node is the *root of the document*, which must be distinguished from the *root element*, its only element child. This somehow misleading vocabulary is part of the price to

pay in order to master the XML data model.

An important notion (related to the physical nature of a document and not to its logical structure) is the notion of *entity*. Examples of entities are as follows:

```
<!ENTITY chap1 "Chapter 1: to be written">
<!ENTITY chap2 SYSTEM "chap2.xml">
<report> &chap1; &chap2 </report>
```

The content of an entity may be found in the document (as entity `chap1`), in the local system (as for `chap2`) or on the Web (with a URI). The content of an entity can be XML. In this case, the document is logically equivalent to the document obtained by replacing the entity references (e.g., `&chap1; &chap2`) by their content. The content of the entity may also be in some other format (e.g., Jpeg). In such case, the entity is not parsed and treated as an attachment.

**Remark 2.2** (Details)

1. An *empty* element `<name></name>` may alternatively be denoted `<name/>`.
2. An element or attribute name is a sequence of alphanumeric and other allowed symbols that must start with an alphanumeric symbols (or an underscore).
3. The value of attribute is a string with certain limitations.
4. An element may also contain comments and processing instructions.

## 2.4 Typing and namespaces

XML documents need not typed. They *may* be. The first kind of typing mechanism originally introduced for XML is DTDs, for *Document Type Definitions*. DTDs are still quite often used. We will study in Chapter **??** *XML schema*, which is more powerful and is becoming more standard, notably because it is used in Web services.

An XML document including a type definition is as follows:

```
<?xml version="1.0" standalone="yes" ?>
<!-- This is a comment - Example of a DTD -->
<!DOCTYPE email [
  <!ELEMENT  email ( header, body )>
  <!ELEMENT header ( from, to, cc? )>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT cc (#PCDATA)>
  <!ELEMENT body (paragraph*) >
  <!ELEMENT paragraph (#PCDATA)>
<email>
  <header>
     <from> af@abc.com </from>
     <to> zd@ugh.com </to>
  </header>
  <body>
```

```
  </body>
</email>
```

The DOCTYPE clause declares the type for this document. Such a type declaration is not compulsory. Ignoring the details of this weird syntax, this is stating, for instance, that a header is composed of a from element, a to one, and possibly a cc one, that a body consists of a list of paragraphs, and finally that a paragraph is a string.

In general, the list of children for a given element name is described using a regular expression in BNF specified for that element.

A most important notion is also that of *namespace*. Consider a label such as job. It denotes different notions for a hiring agency or for a (computer) application service provider. Applications should not confuse the two notions. The notion of namespace is used to distinguish them. More precisely, consider the following XML piece of data:

```
<doc xmlns:hire='http://a.hire.com/schema'
     xmlns:asp='http://b.asp.com/schema'  >
...
<hire:job> ... </hire:job> ...
<asp:job> ... </asp:job> ...
</doc>
```

The hire namespace is linked to a schema, and the asp to another one. One can now mix the two vocabularies inside the same document in spite of their overlap.

XML also provides some referencing mechanisms that we will ignore for now.

When a type declaration is present, the document must conform to the type. This may, for instance, be verified by an application receiving a document before actually processing it. If a well-formed document conforms to the type declaration (if present), we say that it is *valid* (for this type).

## 2.5 To type or not to type

The structure of an XML document is included in the document in its label structure. As already mentioned, one speaks of self-describing data. This is an essential difference with standard databases:

> *In a database, one defines the type of data (e.g., a relational schema) before creating instances of this type (e.g., a relational database). In semistructured data (and XML), data may exist with or without a type.*

The "may" (in *may exist*) is essential. Types are not forbidden; they are just not compulsory and we will spend quite some effort on XML typing. But in many cases, XML data often presents the following characteristics:

1. the data are irregular: there may be variations of structure to represent the same information (e.g., a date or an address) or unit (prices in dollars or euros); this is typically the case when the data come from many sources;

2. parts of the data may be missing, for example, because some sources are not answering, or some unexpected extra data (e.g., annotations) may be found;

3. the structure of the data is not known a priori or some work such as parsing has to be performed to discover it (e.g., because the data come from a newly discovered source);

4. part of the data may be simply untyped, (e.g., plain text).

Another differences with database typing is that the type of some data may be quite complex. In some extreme cases, the size of the type specification may be comparable to, or even greater than, the size of the data itself. It may also evolve very rapidly. These are many reasons why the relational or object database models that propose too rigid typing were not chosen as standards for data exchange on the Web, but a semistructured data model was chosen instead.

# 3 Web Data Management with XML

XML is a very flexible language, designed to represent contents independently from a specific system or a specific application. These features make it *the* candidate of choice for data management on the Web.

Speaking briefly, XML enables *data exchange* and *data integration*, and it does so *universally* for (almost) all the possible application realms, ranging from business information to images, music, biological data, and the like. We begin with two simple scenarios showing typical distributed applications based on XML that exploit exchange and integration.

## 3.1 Data exchange

The typical flow of information during XML-based data exchange is illustrated on Figure 6. Application *A* manages some internal data, using some specialized data management software, (e.g., a relational DBMS). Exchanging these data with another application *B* can be motivated either for publication purposes, or for requiring from *B* some specialized data processing. The former case is typical of *web publishing* frameworks, where *A* is a *web server* and *B* a web client (browser, mobile phone, PDF viewer, etc.). The later case is a first step towards distributed data processing, where a set of sites (or "peers") collaborate to achieve some complex data manipulation.

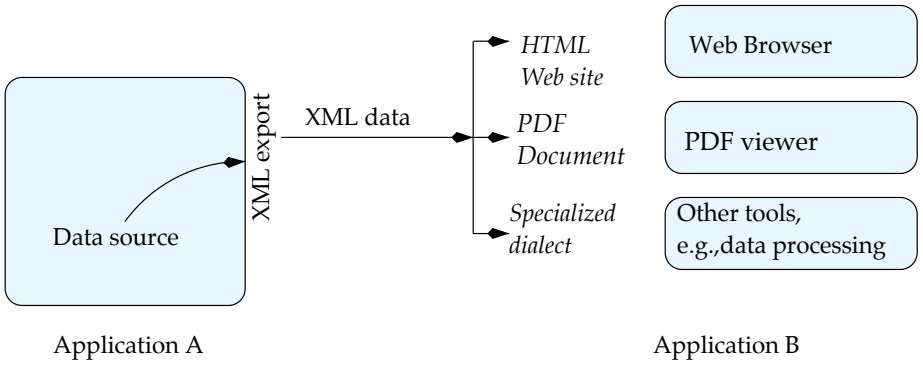



Figure 6: Flow of information in XML-based data exchange

XML is at the core of data exchange. Typically, *A* first carries out some conversion process (often called "XML publishing") which produces an appropriate XML representation from the internal data source(s.) These XML data are then consumed by *B* which extracts the content, processes it, and possibly returns an XML-encoded result. Several of the afore mentioned features of XML contribute to this exchange mechanism:

1. ability to represent data in a serialized form that is safely transmitted on the Web;
2. typing of document, which allows *A* and *B* to agree on the structure of the exchanged content;
3. standardized conversion to/from the serialized representation and the specific tree-based representation respectively manipulated by *A* and *B*.

For concreteness, let us delve into the details of a real Web Publishing architecture, as shown in Figure 7. We are concerned with an application called *Shows* for publishing information about movie showings, in a Web site and in a Wap site. The application uses a relational database. Data are obtained from a relational database as well as directly from XML files. Some specialized programs, written with XSLT (the XML transformation language, see below) are used to restructure the XML data, either coming from files, from the database through a conversion process, or actually from any possible data source. This is the *XML publishing* process mentioned above. It typically produces XHTML pages for a Web site. These pages are made available to the world by a Web server.

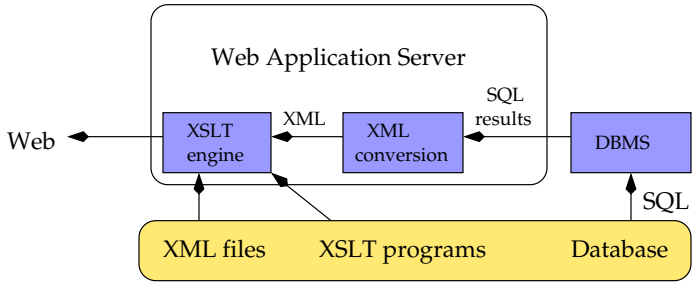



Figure 7: Software architecture for Web publishing applications

The data flow, with successive transformations (from relational database to XML; from XML to a publication dialect), is typical of XML-based applications, where the software may be decomposed in several modules, each dedicated to a particular part of the global data processing. Each module consumes XML data as input and produces XML data as output, thereby creating chains of data producers/consumers. Ultimately, there is no reason to maintain a tight connection of modules on a single server. Instead, each may be hosted on a particular computer somewhere on the Internet, dedicated to providing specialized services.

## 3.2 Data integration

A typical problem is the integration of information coming from heterogeneous sources. XML provides some common ground where all kinds of data may be integrated. See Figure 8. For each (non-XML) format, one provides a *wrapper* that is in charge of the mapping from the

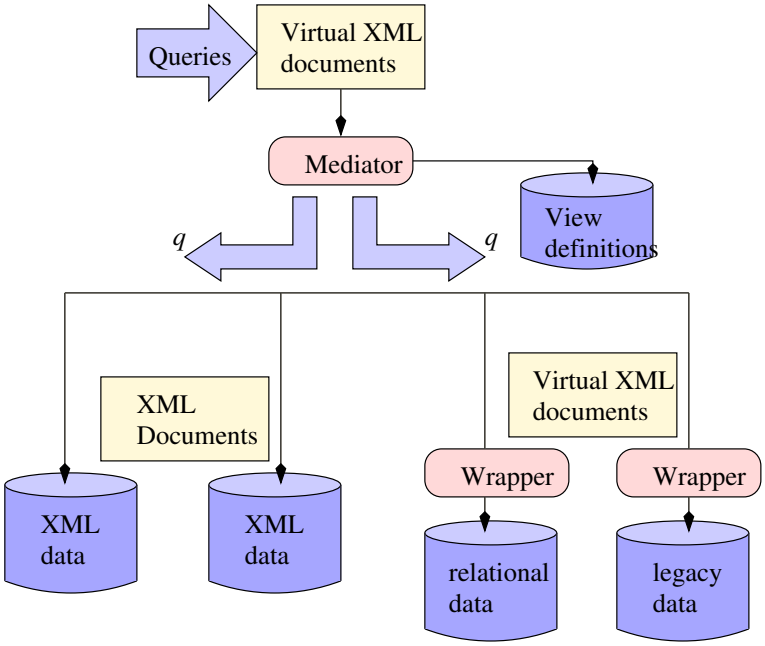



Figure 8: Information integration

world of this format to the XML world. Now a query (say an XQuery) to the global XML view is transformed by the mediator (using the view definitions) into queries over the local sources. A source wrapper translates the XML query to the source it receives into a query the source understands. That query is evaluated on the source, and some data are produced. The wrapper translates this data into XML data. The mediator combines the result it receives from all the wrappers to obtain the general result.

# 4 The XML World

The broad scope of XML is achieved through a spectrum of XML dialects, XML-based standards, and XML-based software. Dialects define specialized structures, constraints, and vocabularies to construct ad hoc XML contents that can be used and exchanged in a specific application area. Languages and softwares on the other hand are *generic*. Together, dialects and languages build an entire world that is at our disposal for developing Web applications.

## 4.1 XML dialects

Suppose we are working in some particular area, say the industry of plastic. To facilitate the exchange of information, the industry specifies a common type for such exchanges, with the tags that should be used, the structure of the information they contain, and the meaning of the corresponding data. The advantage is that once this is achieved, (i) partners can easily exchange information, (ii) information from different companies can more easily be integrated, and (iii) information of interest can more easily be found. Basically, by doing that, the plastic industry has solved, in part, the problem of the heterogeneity of information sources. It is

important to note that the design of such dialect includes the design of a syntax (an XML type) and of a semantics (e.g., the meaning for the different element of the syntax).

We already mentioned the XHTML that serves the same purpose as HTML (describe simple documents) but with an XML syntax. Perhaps the main difference is that all opening tags should be closed. RSS is another popular dialect for describing content updates that is heavily used for blog entries, news headlines, or podcasts. The following document is an example of RSS content published on the WebDam site (`http://webdam.inria.fr/`):

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

<rss version="2.0">

<channel>
        <title>Webdam Project</title>
        <atom:link href="http://webdam.inria.fr/wordpress/?feed=rss2"
                    rel="self" type="application/rss+xml" />
        <link>http://webdam.inria.fr/wordpress</link>
        <description>Foundations of Web Data Management</description>
        <pubDate>Wed, 26 May 2010 09:30:54 +0000</pubDate>

  <item>
    <title>News for the beginning of the year</title>
    <description>The webdam team wish you an happy new year!</description>
    <link>http://webdam.inria.fr/wordpress/?p=475</link>
    <pubDate>Fri, 15 Jan 2010 08:48:45 +0000</pubDate>
    <dc:creator>Serge</dc:creator>
    <category>News</category>
  </item>

</channel>
</rss>
```

SVG (Scalable Vector Graphics) is an XML dialect for describing two-dimensional vector graphics, both static and animated. SVG is very powerful and can represent quite complex figures such as all the figures found in the present book! The following is a simple example that shows the combination of a surfacic object with some text. The left part is the SVG encoding, the right one shows the graphic representation that can be obtained by a Web browser or by a specialized tool (e.g., Gimp or Inkscape).

```
<?xml version="1.0" encoding="UTF-8" ?>
<svg xmlns="http://www.w3.org/2000/svg">

   <polygon points="0,0  50,0  25,50"
        style="stroke:#660000; fill:#cc3333;"/>

  <text x="20" y="40">Some SVG text</text>
</svg>
```

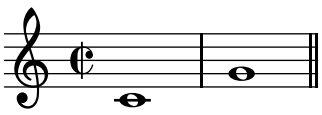
This example shows that data of any kind can be encoded as XML, and exchanged on the Internet between applications that possibly run under different systems, on different computer architectures, and so on. It is also worth noting that, although this SVG example is trivial and easy to understand even without a rendering tool, in general the content of an XML file may be arbitrarily complex and definitely not suitable for inspection by a human being. Some of the SVG representations for complex figures in this book consist of hundreds of lines of abstruse code that can only be manipulated via appropriate software.

As another illustration, (symbolic) music can be represented in XML. The following is a slightly simplified example of a MusicXML document.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<score-partwise version="2.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
      <attributes>
        <divisions>1</divisions>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
      </note>
      <note>
        <pitch>
          <step>G</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
      </note>
  </part>
</score-partwise>
```

This encoding can be interpreted by specialized software and rendered as a musical score:



Some other popular XML dialects are MathML (the mathematical mark-up language), an XML dialect for describing mathematical notation and capturing both its structure and content. It aims at integrating mathematical formulae into World Wide Web documents (if one considers only the presentation aspect, it is something like the mathematics in LaTeX): see Exercises. XML/EDI is an XML dialect for business exchanges. It can be used to describe, for

instance, invoices, healthcare claims, and project statuses. For the moment, the vast majority of electronic commerce transactions in the world are still not using XML, but (pure) EDI, a non-XML format.

There are just too many dialects to mention them all, ranging from basic formats that represent the key/value configuration of a software (look at your Firefox personal directory!) to large documents that encode complex business process. Above all, XML dialects can be created at will, making it possible for each community to define its own exchange format.

## 4.2 XML standards

The universality of XML brings an important advantage: any application that chooses to encode its data in XML can benefit from a wide spectrum of standards for defining and validating types of documents, transforming a document from one dialect to another, searching the document for some pattern, manipulating the document via a standard programming language, and so on. These standards are generic to XML, and are defined independently from the specificities of a particular dialect. This also enables the implementation of softwares and languages that are generic, as they apply to XML-formatted information whatever the underlying application domain. For the standards, one should also notably mention:

**SAX,** the Simple API for XML, is an application programming interface (API) providing a serial access to XML documents seen as a sequence of tokens (its serialization).

**DOM,** the Document Object Model, is an object-oriented model for representing (HTML and) XML document, independently from the programming language. DOM sees a document as a tree and provides some navigation in it (e.g., move to parent, first child, left/right sibling of a node). A DOM API is available for all popular languages (Java, C++, C#, Javascript, etc.)

**XPath,** the XML Path Language, is a language for addressing portions of an XML document.

**XQuery** is a flexible query language for extracting information from collections of XML documents. It is to a certain extent the SQL for Web data.

**XSLT,** the Extensible Stylesheet Language Transformations, is a language for specifying the transformation of XML documents into other XML documents. A main usage of XSLT is to define *stylesheet* to transform some XML document into XHTML, so that it can be displayed as a Web page.

**Web services,** provide interoperability between machines based on Web protocols. See further.

To make the discussion a bit more precise, we consider some of these in slightly more detail.

### Programming interfaces: SAX and DOM

We start with the first two APIs, that provide two distinct ways to see an XML document. See Figure 9.

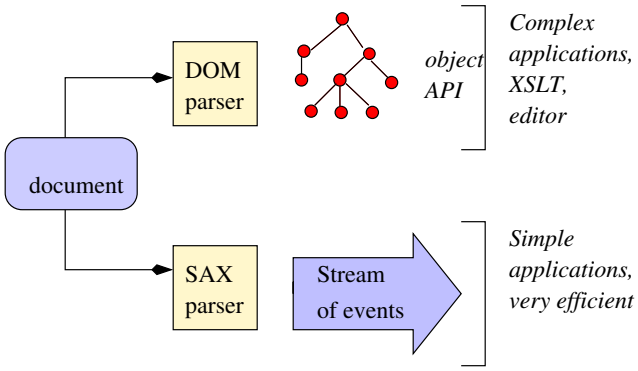



Figure 9: Processing an XML document with SAX and DOM

Let us begin with the SAX programming model. A SAX parser transforms an XML document into a flow of events. Examples of events are the start/end of a document, the start/end of an element, a text token, a comment, and so on. To illustrate, suppose that we obtained some relational data in an XML format. SAX may be used, for instance, to load this data in a relational database as follows:

1. when document start is received, connect to the database;
2. when a *Movie* open tag is received, create a new *Movie* record;
   (a) when a text node is received, assign its content to $X$;
   (b) when a *Title* close tag is received, assign $X$ to *Movie.Title*;
   (c) when a *Year* close tag is received, assign $X$ to *Movie.Year*, etc.
3. when a *Movie* close tag is received, insert the *Movie* record in the database (and commit the transaction);
4. when document end is received, close the database connection.

SAX is a good choice when the content of a document needs to be examined once (as in the previous example), without having to follow some complex navigation rule that would, for instance, require to turn back during the examination of the content. When these conditions are satisfied, SAX is the most efficient choice as it simply scans the serialized representation. For concreteness, the following piece of code shows a SAX handler written in Java (this example is simplified for conciseness: please refer to the Web site for a complete version). It features methods that handle SAX events: opening and closing tags; character data.

```
import org.xml.sax.*;
import org.xml.sax.helpers.LocatorImpl;

public class SaxHandler implements ContentHandler {

    /** Constructor */
    public SaxHandler() {
```

```
        super();
    }

    /** Handler for the beginning and end of the document */
    public void startDocument() throws SAXException {
        out.println("Start the parsing of document");
    }

    public void endDocument() throws SAXException {
        out.println("End the parsing of document");
    }

    /** Opening tag handler */
    public void startElement(String nameSpaceURI, String localName,
        String rawName, Attributes attributes) throws SAXException {
        out.println("Opening tag: " + localName);

        // Show the attributes, if any
        if (attributes.getLength() > 0) {
            System.out.println("  Attributes:  ");
            for (int index = 0;
                    index < attributes.getLength(); index++) {
                out.println("     - " + attributes.getLocalName(index)
                   + " = " + attributes.getValue(index));
            }
        }
    }

    /** Closing tag handler */
    public void endElement(String nameSpaceURI,
                    String localName, String rawName)
        throws SAXException {
        out.print("Closing tag : " + localName);
        out.println();
    }

    /** Character data handling */
    public void characters(char[] ch, int start,
                    int end) throws SAXException {
        out.println("#PCDATA: " + new String(ch, start, end));
    }
}
```

The other XML API is DOM. A DOM parser transforms an XML document into a tree and, as already mentioned, offers an object API for that tree. A partial view of the class hierarchy of DOM is given in Figure 10. We give below a *Preorder* program that takes as argument the name of some XML file and analyzes the document with a DOM parser. The analysis traverses the XML tree in preorder and outputs a message each time an **element** is met. Comments in the code should clarify the details.

```
// Import Java classes
```

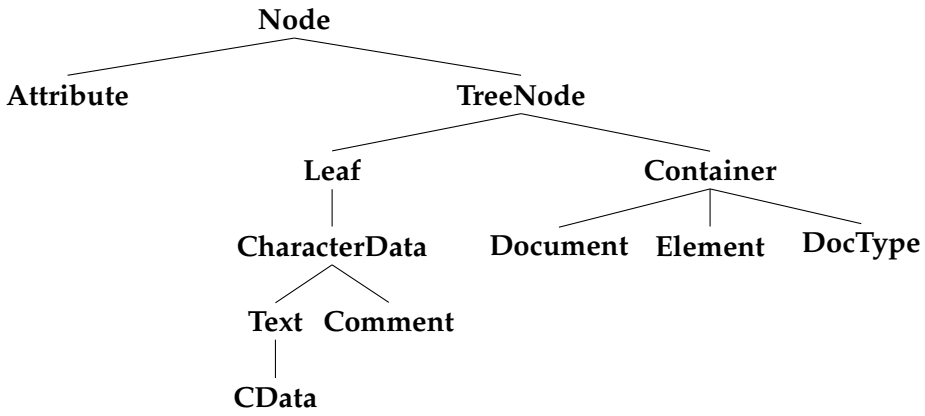



Figure 10: DOM class hierarchy

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

/**
 * A DOM class that outputs all the elements in preorder
 */

class DomExample {
  /**
   * The main method.
   */
  public static void main(String args[]) {
      // Obtain the document
      File fdom = new File(args[0]);

      // Parser instantiation
      DocumentBuilderFactory factory =
          DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = factory.newDocumentBuilder();

      // Document analysis
      Document dom = builder.parse(fdom);

      // Start the recursive traversal from the root element
      Node elementRoot = dom.getDocumentElement();
      Traversal(elementRoot);
  }

  /**
   * The recursive method.
   */
  private static void Traversal(Node node) {
    String str = new String();
    // Node numbering if it is a text
    if (node.getNodeType() == Node.ELEMENT_NODE) {
```

```
      str = "Found element " + node.getNodeName();
      System.out.println(str + "\n");

      // Recursive call if node has children
      if (node.hasChildNodes()) {
        // Get the list of children
        NodeList child = node.getChildNodes();
        // List traversal
        for (int i = 0; i < child.getLength(); i++)
          Traversal(child.item(i));
      }
    }
  }
}
```

Several implementations of DOM exist. The example we use here is based on an implementation proposed by the Apache Foundation and a popular DOM parser called Xerces.

**Query languages: XPath, XQuery**

Consider a large collection of XML documents, say the archives of the DBLP bibliography. To extract some pieces of information from this collection, a user will typically specify graphically a query. That query may be translated in XPath or XQuery queries in much the same way that a standard graphical query to a relational database is translated to SQL.

In an XML context, queries combine different styles of selection:

1. queries by keywords as in search engines;
2. precise queries as in relational systems;
3. queries by navigation as in Web browsing.

Loosely speaking, XPath is a language that allows the specification of paths between the nodes of an XML document (seen as a tree, as it always should). This specification takes the form of *patterns* that describe more or less tightly a family of paths that comply to the specification. These paths "match" the pattern. An example of XPath pattern query is as follows:

```
document('dblp.xml')//book[publisher = 'Cambridge University Press']
```

It selects the books in the document *dblp.xml* with *Cambridge University Press* for publisher. XPath is at the core of other XML manipulation languages, such as XQuery and XSLT, because it provides a mechanism to navigate in an XML tree.

Here is an example of query with XQuery.

```
for $p in document('dblp.xml')//publisher
let $b := document('dblp.xml')//book[publisher = $p]
where count($b) > 100
return <publisher> {$p//name, $p//address} </publisher>
```

In this query, the variable $\$p$ scans the list of publishers. For each publisher, variable $\$b$ contains the sequence of books published by this publisher. The **where** clause filters out the publishers who published less than 100 books. Finally, the **return** constructs the result, for each publisher, the name and address.

### Web services

An application on a machine when turned into a Web service can be used by a remote machine. This is the basis of distributed computing over the Internet. Different machines over the network exchange XML data using a particular protocol, *SOAP*. They describe their interfaces using yet another language, namely *WSDL* (pronounced wiz-d-l), the Web Services Description Language.

The idea underlying Web services is very simple and will be best explained by an example. Suppose I wrote a program that takes as input a URL and computes the page rank of that page and its classification in some ontology (what it is talking about). Suppose a friend in California wants to use my program. I have to package it, send her all the data (perhaps some databases) the program is using (which may be forbidden by my company). Then we have to solve loads of problems such as software compatibility. It is much simpler to turn my program into a Web service (which takes a couple of minutes) and publish it on a local Web server. My friend can then use it without knowing that I developed it in Java or C++, on Mandrake Linux or Vista, with standard libraries or rather obscure homemade ones.

The core ideas are to exchange (serialized) XML and use a standard protocol for messages. The basis is SOAP, the Simple Object Access Protocol, a protocol for exchanging XML-based messages over the network (typically using HTTP or HTTPS). The most common messaging for SOAP is a Remote Procedure Call (RPC) where a computer (the client) sends a request message to another one (the server); and the server responds by a message to the client. Imagine for instance that you make the following function call from your Client application:

```
pr = getPageRank ("http://webdam.inria.fr/");
```

This call must be shipped to the server. The SOAP protocol encodes the relevant information in XML and transfers the following XML document to the server.

```
<?xml version="1.0" encoding="UTF-8">
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <SOAP-ENV:Body>
              <ns1:getPageRank
               xmlns:ns1="urn:PageRankService">
                     <param1 xsi:type="xsd:string">
                        http://webdam.inria.fr/
                     </param1>
              </ns1:getPageRank>
      </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

Although rather verbose, this SOAP message is simple enough to exhibit the main information that constitutes the remote function call: the server Uniform Resource Name (urn), the function name and the parameter value. The server then transmits its answer with a SOAP message. This exchange is transparent to the Client: what is exploited here is the capacity of XML to safely encode and exchange data between computers.

Let us finally complete this very brief overview of Web Services by mentionning WSDL, the Web Services Description Language. WSDL is an XML-based language for describing Web services, which specifies the type of their input and output. It can be used, in particular, to generate automatically the correct "stubs" of client applications that takes care of creating the SOAP message that respects the signature (type and number of parameters) of the functions provided by the service.

## 5 Further reading

Before the Web, publication of electronic data was limited to a few scientific and technical areas. With the Web and HTML, it rapidly became universal. HTML is a format meant for presenting documents to humans. However, a lot of the data published on the Web is produced by machines. Moreover, it is more and more the case that Web data are consumed by machines as well. Because HTML is not appropriate for machine processing, *semistructured data models*, and most importantly of a new standard for the Web, namely XML [W3C08], were developped in the 90'. The use of a semistructured data model as a standard for *data representation* and *data exchange* on the Web brought important improvement to the publication and reuse of electronic data by providing a simple syntax for data that is machine-readable and, at the same time, human-readable (with the help of the so-called style-sheets).

Semistructured data models may be viewed, in some sense, as bringing together two cultures that were for a long while seen as irreconcilable, document systems (with notably SGML [Gol90]) and database systems (with notably relational systems [Ull88]). From a model perspective, there are many similarities with the object database model [Cat94]. Indeed, like XML, the object database model is based on trees, provides an object API, comes equipped with query languages and offers some form of serialization. As already mentioned, an alternative to XML in some contexts is JSON (see `http://www.json.org` and the description in Chapter **??**), a semistructured model directly derived from the need to serialize the representation of an object that must be exchanged by two programs (typically, a Web browser and a Web server). A main difference is that the very rigorous typing of object databases was abandoned in semistructured data models.

SGML (Standard Generalized Markup Language) is the (complex) 1986 ISO Standard for data storage and exchange. SGML dialects can be defined using DTD. For instance, HTML is such a dialect.

XML is developed and promoted by the World Wide Web Consortium (W3C). XML is a 1998 recommendation of the W3C. Its specification is a couple of dozens of pages long, vs. the hundreds of pages of SGML. It is supported by academic labs such as MIT (US), INRIA (Europe) or Keio University and backed by all the heavy weights of industry notably Oracle, IBM and Microsoft. The role of W3C is in particular to lead the design of standards where the XML syntax is only the tip of the iceberg. They propose a wide range of them for typing

XML [W3C04], querying XML [XQu], transforming XML [XSL], interacting with XML [DOM], developing distributed applications with XML, etc. See the site of the W3C [W3C] for more.

The articulation of the notion of semistructured data may be traced to two simultaneous origins, the OEM model at Stanford [PGMW95, AQM+97] and the UnQL model at U. Penn [PDS95]. See [ABS99] for a first book on the topic.

Specific data formats had been previously proposed and even became sometimes popular in specific domains, e.g. ASN.1 [ISO87]. The essential difference between data exchange formats and semistructured data models is the presence of high level query languages in the latter. A query language for SGML is considered in [ACC+97]. Languages for semistructured data models such as [AQM+97, PDS95] then paved the way for languages for XML [XQu].

# 6 Exercises

## 6.1 XML documents

**Exercise 6.1 (Well formed XML documents)** *Have you ever written an HTML page? If not, it is time to create your first one: create a* `.html` *home page with your public information: CVs, address, background and hobbies, photos, etc.*

*This page* must *be a well-formed XHTML document. Use a public XHTML validator to check its well-formedness, and correct any error.* Hints: *the W3C provides an online validator at http://validator.w3c.org/. You can also add a validator to your browser that check any page loaded from the Internet (for Firefox, the Web Developper plugin is a good choice).*

**Exercise 6.2 (XML and graphics)** *Now, embellish you page with some vector graphics. As a starting point, take the SVG example given in the present chapter, save it in an svg.xml document and add the following instruction somewhere in your XHTML code.*

```
<object data="svg.xml" type="image/svg+xml" width="320" height="240" />
```

*Open the page in your browser (of course, the browser should be equipped with an SVG rendering module: Firefox natively supports SVG) and see the graphics displayed in your Web page. Search for some more exciting SVG options and experiment them.*

**Exercise 6.3** *MathML is an XML dialect for the representation of mathematical fomulas in XML. Arithmetic formulas in MathML use a* prefix *notation, where operators come before their operands. For instance, the prefix notation of*

$$x^2 + 4x + 4$$

*is*

```
(+ (^ x 2) (* 4 x) 4)
```

*When encoded in MathML, this formula is represented by the following document:*

```
<?xml version='1.0'?>
<apply>
  <plus/>
```

```
  <apply>
    <power/>
     <ci>x</ci>
     <cn>2</cn>
   </apply>
   <apply>
     <times/>
     <cn>4</cn>
     <ci>x</ci>
   </apply>
   <cn>4</cn>
</apply>
```

*Note that each parenthesis gives rise to an* `apply` *element ; operators* $+$, $\times$ *and* `^` *are respectively represented with* `plus`, `times` *and* `power` *elements ; finally, variables are represented by* `ci` *elements, and constants* `cn` *elements.*

1. *Give the tree for of this MathML document.*
2. *Express the following formulas in MathML*
   - $(x^y + 3xy) \times y$
   - $x^{a+2} + y$
3. *Give the DTD that corresponds to the MathML fragment given above.*

## 6.2 XML standards

Programming with XML APIs, SAX and DOM, is a good means to understand the features of XML documents. We invite you to realize a few, simple programs, based on examples supplied on our web site.

These examples are written in Java. You need a SAX/DOM parser: the Xerces open-source parser is easy to obtain and our programs have been tested with it:

- get the Xerces java archive from `http://xerces.apache.org/` and download it sowewhere on your local disk;
- add your Xerces directory to `$JAVA_HOME`;
- take from our web site the following files: *SaxExample.java*, *SaxHandler.java* and *DomExample.java*.

Let us try the SAX program first. It consists of a class, the *handler*, that defines the method triggered when syntactic tokens are met in the parsed XML document (see page 19 for details). The handler class is supplied to the *SAX parser* which scans the XML document and detects the tokens. Our handler class is in *SaxHandler.java*, and the parser is instantiated and run in *SaxExample.java*. Look at both files, compile them an run *SaxExample*. It takes as input the name of the XML document. For instance, using the *movies.xml* document from our site:

```
java SaxExample movies.xml
```

The DOM example executes the same basic scan of the XML document in preorder, and outputs the name of each element. Compile it, and run it on the same file:

```
java DomExample movies.xml
```

We also provide a *DomPreorder.java* example that shows a few other features of DOM programming: modification of nodes, and serialisation of a DOM object.

For the following exercises, you should download the *dblp.xml* document from the DBLP site: `http://www.informatik.uni-trier.de/~ley/db/`. The main file is about 700 Mbs, which helps to assess the respective performance of the SAX and DOM approaches.

**Exercise 6.4 (Performance)** *Write a SAX program that count the number of top-level elements (elements under the element root) in an XML document.*

- *apply your program to dblp.xml and count the number of references;*
- *extend your program to count only a subset of the top-level elements, say, journals or books.*

*Write the same program as above, but in DOM. Run it on dblp.xml and compare the performances.*

**Exercise 6.5 (Tree-based navigation)** *Imagine that you need to implement a Navigate program that accesses one or several nodes in an XML documents, referred to by a path in the hierarchy. For instance:*

```
java Navigate movies movie title
```

*should retrieve all the* `<title>` *nodes from the movies.xml document (nb: this is actually a quite rudimentary XPath evaluator, see the next chapter).*

*Try to design and implement this program in SAX and DOM. Draw your conclusions.*


[ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan-Kaufman, 1999.

[ACC+97] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and Jerome Simeon. Querying documents in object databases. *Intl. Journal on Digital Libraries*, 1:5–19, 1997.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Intl. Journal on Digital Libraries*, 1:68–88, 1997.

[Cat94] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1994.

[DOM] Document Object Model. w3.org/DOM.

[Gol90] C.F. Goldfarb. *The SGML Handbook*. Calendon Press, Oxford, 1990.

[ISO87] ISO. Specification of astraction syntax notation one (asn.1), 1987. Standard 8824, Information Processing System.

[PDS95] P.Buneman, S. Davidson, and D. Suciu. Programming constructs for unstructured data. In *Proc. Intl. Workshop on Database Programming Languages (DBLP)*, 1995.

[PGMW95] Y. Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *Proc. Intl. Conf. on Data Engineering (ICDE)*, 1995.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge Base Systems, Volume I*. Computer Science Press, 1988.

[W3C] World wide web consortium. http://www.w3.org/.

[W3C04] W3C. XML Schema Part 0: Primer. `http://www.w3.org/TR/xmlschema-0/`, October 2004.

[W3C08] W3C. Extensible markup language (XML) 1.0. `http://www.w3.org/TR/REC-xml/`, November 2008.

[XQu] XML Query (XQuery). `http://www.w3.org/XML/Query`.

[XSL] The Extensible Stylesheet Language Family. http://www.w3.org/Style/XSL.