# 1

# Data Model

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.

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

## 1.1 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 3). 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 1.1 and 1.2. In Figure 1.1, the label structure is captured by tree edges, whereas data values reside at leaves. In Figure 1.2, 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 }
```

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
```

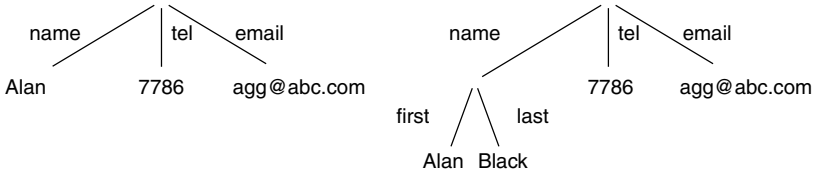



**Figure 1.1.** Tree representation, with labels on edges.

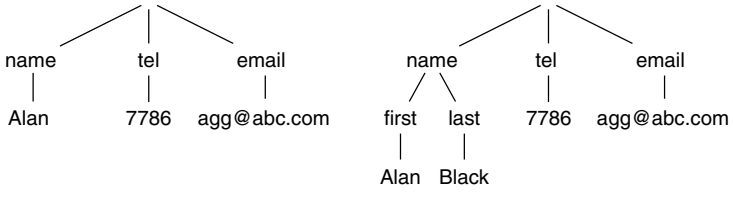



**Figure 1.2.** Tree representation, with labels as nodes.

```
            {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 Section 20.3.

## 1.2 XML

XML, the eXtensible Mark-up 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 of transmission on the Internet.

**Remark 1.2.1** 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, XSchema, DOM, SAX, Xpath, XSL, XQuery, SOAP, WSDL, and so forth.

### 1.2.1 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 1.3 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

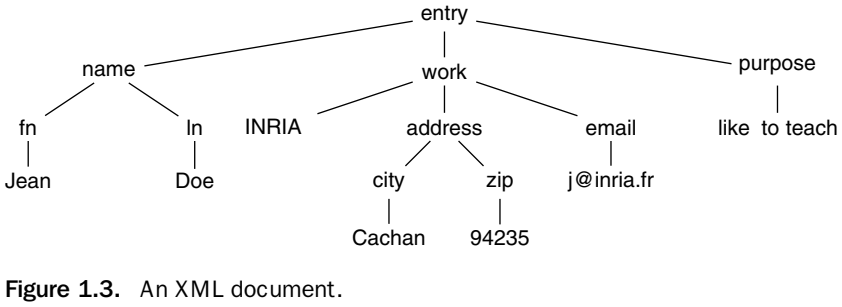



**Figure 1.3.** An XML document.

```
<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,  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  closing tags.)

The presentation of that HTML document by a classical browser can be found in Figure 1.4. 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

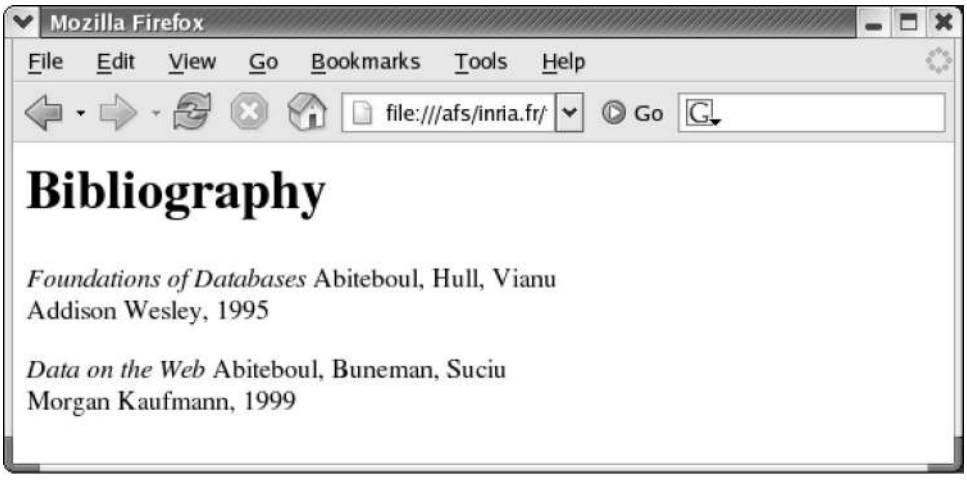



**Figure 1.4.** HTML presentation.

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.

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.

### 1.2.2 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 prerequisite 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 1.5 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.

Stricly speaking, the *syntax* of XML relates to its serialized representation. The syntax can be normalized because a serialized document is meant for data

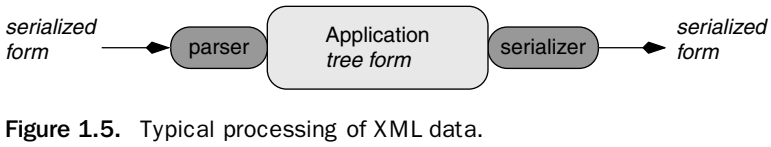



**Figure 1.5.** Typical processing of XML data.

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 if defined by the World Wide Webb 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 Section 1.2.1).

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.

### 1.2.3 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, and Chinese, can be represented. An element is of the form

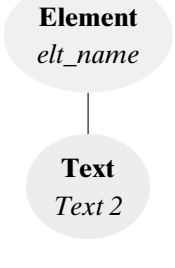
```
<name attr='value' ... > content </name>
```

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:

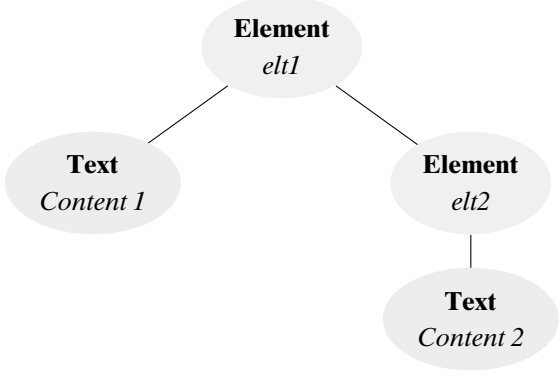



The **Element** nodes are the internal nodes of a DOM representation. They represent the hierarchical structure of the document. An **Element** node has a *name*, which is the label of the corresponding tag in the serialized form. The second type of node illustrated by this example is a **Text** node.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

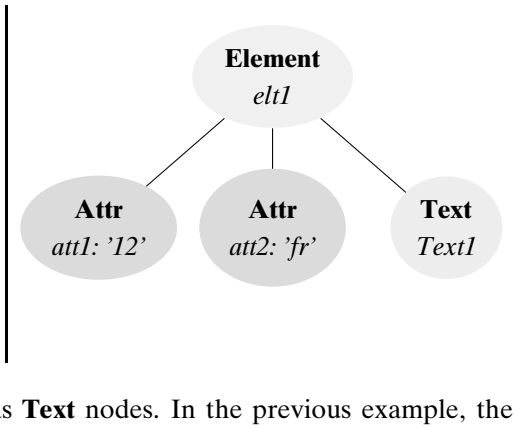
```
<report language='fr' date='08/07/07'>
```

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.