Informatics 1
School of Informatics, University of Edinburgh

Data and Analysis

Part II
Semistructured Data

Ian Stark
February 2011

Part II — Semistructured Data

XML:

Corpora:

Recommended reading

[DMS], pp. 227–231, covers the topic, but rather superficially.

For a more in-depth treatment see Chapter 2 of:

[XWT] An Introduction to XML and Web Technologies
A. Møller and M. Schwartzbach
Addison Wesley, 2006

"A superb summary of the main Web technologies. It is broad and deep giving you enough detail to get real work done. Eminently readable with excellent examples and touches of humour. This book is a gem."

Prof. Philip Wadler, University of Edinburgh

Background

Relational databases record data in tables conforming to relational schemata. This imposes a particular kind of rigid structure on data.

In many situations, it is useful to structure data in a less rigid way; for example:

- when the data has no strong inherent structure; or there is structure, but it varies from item to item;
- when we wish to *mark up* (i.e. annotate) existing unstructured data (e.g. text) with additional information (e.g. semantic annotations);
- when the structure of the data changes over time, perhaps as more data accumulates.

Semistructured data

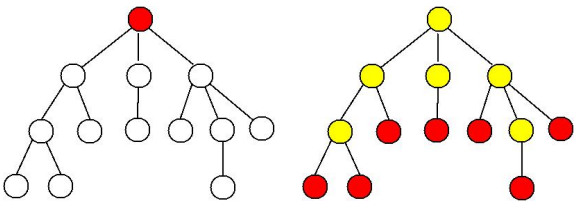
Even *semistructured data* does still impose some structure on data. This generally takes the form of a *tree*.

Before seeing how trees are used in semistructured data, we review basic terminology for talking about trees (the mathematical structure, not the vegetation).

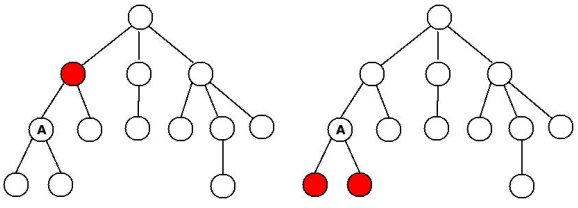
A *tree* structure consists of a set of *nodes*, amongst which there is a unique *root node*. For every node in the tree, there is a unique path from the root node to the node.

Nodes separate into two disjoint classes: *leaves* and *internal nodes*.

Every node other than the root has a unique *parent* node. Every internal node has a nonempty set of *children* nodes. Any two nodes with the same parent are *siblings*.



Root node Leaves and internal nodes



Parent of A Children of A

Semistructured data models

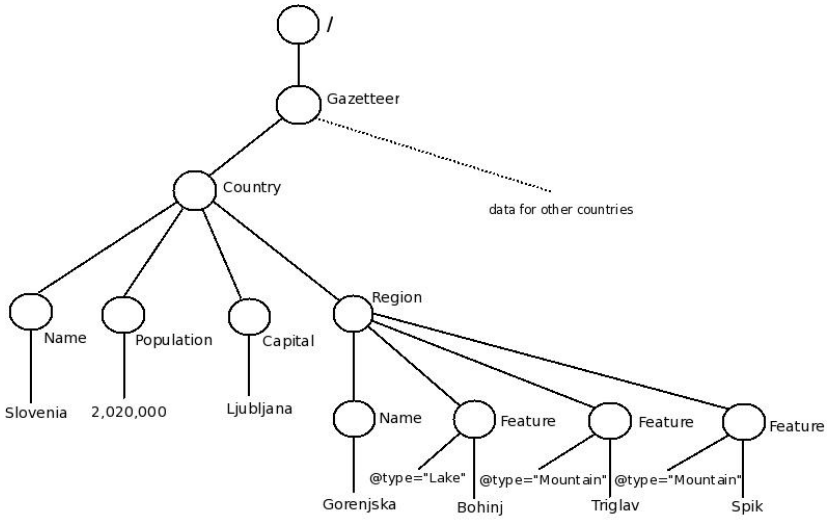
Data is incorporated into a tree structure using a *semistructured data model*.

There are several different such data models.

We shall use the *XPath data model*, selected because its structure corresponds exactly to that of XML.

The next slide illustrates an example of data structured according to the XPath data model.

The example is a fragment of a geographical directory, chosen because it readily fits in a hierarchical tree-based structure.



Types of node in the XPath data model

Root node. This is the root of the tree. It is labelled **`/`**.

Element nodes. These are nodes labelled with *element names*, which serve the purpose of categorising the data below them. In the example, the element names are: **`Gazetteer`**, **`Country`**, **`Name`**, **`Population`**, **`Capital`**, **`Region`**, and **`Feature`**. In the XPath data model, internal nodes other than the root are always element nodes.

The root node is required to have a single element node as child, called the *root element* (since it is root in the tree of all element nodes). In the example, the root element is **`Gazetteer`**.

Text nodes. These are leaves of the tree where textual information is stored. In the example, the text strings **`"Slovenia"`**, **`"2,020,000"`**, **`"Ljubljana"`**, **`"Gorenjska"`**, **`"Triglav"`**, **`"Bohinj"`** and **`"Špik"`** appear at text nodes.

Attribute nodes

Attribute nodes are leaves of the tree in which an *attribute* associated with the parent element node is assigned a value. In the example, we use the `@` symbol to identify attributes. There is a single attribute **`type`**, it is associated with the **`Feature`** element, and it is assigned the text values **`"Lake"`** and **`"Mountain"`**.

In the XPath data model, attribute nodes are treated differently from other nodes.

Although the parent of an attribute node is an element node, when we talk about the children of this parent node, attribute nodes are not considered to be amongst them.

Since this can be confusing, explicit warnings will be given in situations in which confusion might arise.

Understanding the tree

The meaning of the data at a text node depends on the element nodes that appear along the path from the root of the tree to the leaf, and on the values of the attributes to this node.

For example, the path to **`Bohinj`** is

`/Gazetteer/Country/Region/Feature/`

and the value of the **`type`** attribute of the associated **`Feature`** element is **`"Lake"`**. This tells us that Bohinj is a feature in a region in a country in the gazetteer, and that the type of feature is a lake.

Note that to get further information (such as the name of the country, Slovenia), we need to extract it by following another path from the relevant ancestor element (in this case, the **`Country`** element).

Similarly, the meaning of an element node depends on the path to the node from the root of the tree.

For example, the element **`Name`** is used in two different ways.

A path **`/Gazetteer/Country/Name/`** leads to a text node containing the name of a country.

A path **`/Gazetteer/Country/Region/Name/`** leads to a text node containing the name of a region.

XML is a text-based language for presenting exactly the same tree-structured information as the XPath data model.

XML: Extensible Markup Language

This is a *markup language*, that is it provides a mechanism, based on *elements* (also called *tags*), for annotating (*marking up*) ordinary text with additional information.

It was developed in the mid 1990's from the Standard General Markup Language (SGML) and Hypertext Markup Language (HTML).

XML has a simple text-based format which is convenient for automatically generating and parsing data files, for communicating between programs, and making data available over the web. It is moderately human-readable. XML has become the *de facto* standard for publishing data on the web.

The next slide presents the gazetteer example in XML format.

The content and structure are identical to that of the tree presented earlier. Only the format is different.


```
<Gazetteer>
  <Country>
    <Name>Slovenia</Name>
    <Population>2,020,000</Population>
    <Capital>Ljubljana</Capital>
    <Region>
      <Name>Gorenjska</Name>
      <Feature type="Lake">Bohinj</Feature>
      <Feature type="Mountain">Triglav</Feature>
      <Feature type="Mountain">Špik</Feature>
    </Region>
  </Country>
  <!-- data for other countries here -->
</Gazetteer>
```


XML Elements

Elements (also called *tags*) are the building blocks of XML documents.

The start of the content of an element ***elm*** is marked with the *start tag* **<*elm*>**, and the end of the content is marked with the *end tag* **</*elm*>**.

Elements must be *properly nested*. Thus,

```
<Country><Region> ... </Region></Country>
```

is legal, whereas

```
<Country><Region> ... </Country></Region>
```

is illegal.

Elements are case sensitive, so **REGION** would be different from **Region**.

The content of the **Capital** element

```
<Capital>Ljubljana</Capital>
```

is the text string **"Ljubljana"**.

The content of the **Region** element consists of one **Name** element together with three **Feature** elements in sequence.

The *root element* **Gazetteer** encloses all information in the document.

Although there are no such examples in the example document, the content of an element may be empty, e.g.,

```
<elm></elm>
```

Such *empty elements* can be abbreviated using a single hybrid tag:

```
<elm/>
```


Attributes

An element can have descriptive attributes that provide additional information about the element. For example,

```
<Feature type="Mountain"> ... </Feature>
```

sets the attribute **type** of the given **Feature** element to have value **Mountain**.

Note that attribute values are enclosed in quotation marks (either double or single quotes).

It is possible for one element to have several different attributes, with values defined in sequence within the start tag, e.g.

```
<elm attr1="value1" attr2="value2"> ... </elm>
```


Relating XML and the tree model

The existence of a root element together with the proper nesting of elements ensures that every XML document carries a tree structure in a natural way:

- Each element of the XML document corresponds to an individual element node of the tree.
- The root element of the XML document corresponds to the root element (but *not* the root node) of the tree.
- The text content of an individual XML element corresponds to a child text node of the corresponding element node in the tree.
- An attribute definition in an element's start tag corresponds to a child attribute node of the corresponding element node in the tree.

Comments and processing instructions

Comments can be inserted anywhere in an XML document. Comments start with `<!--` and end with `-->`. They can contain arbitrary text apart from the string `--`.

The full XPath data model also contains *comment nodes* which correspond to XML comments. We have do not consider such nodes in our tree model for two reasons:

1. Simplicity.
2. We have included all the types of node that should be used to store data. Comments should instead be used as aids to the interpretation of the data represented.

XML and the XPath data model also allow *processing instructions* to be included. These are beyond the scope of this course.

Unicode

An XML document is a text document written in *Unicode*.

Unicode is a universal code for "text characters", currently supporting around 100,000 different characters.

The Unicode characters contain the standard 128 ASCII characters, but also many, many other characters in use worldwide, from another 92 scripts.

Each character has an assigned *code point*, which is a number between 0 and 1,114,111 inclusive (hexadecimal 0x0–0x10FFF).

The actual representation of Unicode text in memory or "on the wire" depends on a choice of *encoding* of Unicode character sequences as byte streams. The most common encoding is known as UTF-8; others include UTF-16 and UTF-32.

Well-formed documents

An XML document is one containing text that is *well-formed* according to the XML specification. This requires conformance with several technical guidelines, including:

- It starts with an XML declaration. (Our example gazetteer document does not!) A suitable such declaration would be:

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

  This declares the XML version, and states that UTF-8 character encoding is to be used for Unicode. (Not examinable.)
- It has a root element that contains all other elements.
- All elements are properly nested.

As well as these basic requirements on a document, there may be other constraints on format or content which are useful in particular situations.