

# XML

## MPRI 2.26.2: Web Data Management

---

Antoine Amarilli<sup>a</sup>

Friday, December 14th

---

<sup>a</sup>Based on slides from the Webdam book by Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart  
<https://webdam.inria.fr/Jorge/files/slidesdatamodel.pdf>

# Preliminaries

---

# The role of XML

We have seen how Web pages use **HTML**:

- HTML is appropriate for humans: allows sophisticated output and interaction with textual documents and images;
- HTML falls short when it comes to software exploitation of data.

XML describes **content**, and promotes machine-to-machine communication and **data exchange**

- XML is a generic data format, apt to be specialized for a wide range of fields,  
⇒ (X)HTML is a specialized XML dialect for data presentation
- XML simplifies **data integration**, since data from different sources now share a common format;
- XML comes equipped with many **software products, APIs and tools**.

# Semi-structured data model

A data model to represent both regular and irregular data.

**Self-describing data.** The content comes with its own description;  
⇒ contrast with the **relational model**, where schema and content are represented separately.

**Flexible typing.** Data may be typed (i.e., “such nodes are integer values” or “this part of the graph complies to this description”); often no typing, or a very flexible one

**Serialized form.** The XML document is serialized as text, that can be conveniently stored and exchanged.

## Self-describing data

Starting point: **association lists**, i.e., records of label-value pairs.

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

Natural extension: values may themselves be other structures:

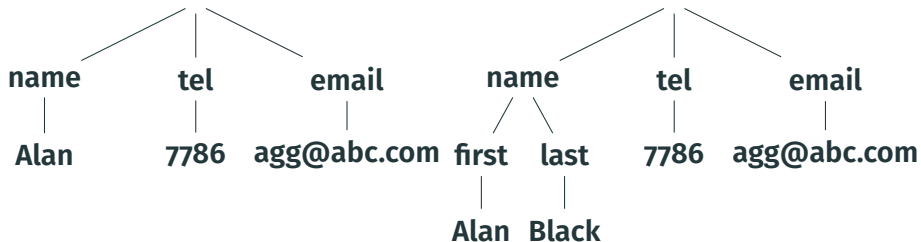
```
{name: {first: "Alan", last: "Black"},  
  tel: 2157786,  
  email: "agb@abc.com"}
```

Further extension: allow duplicate labels.

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

# Tree-based representation

An XML document is represented as a tree with both labels and values as vertices



## Representation of regular data

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"} }
```

→ **Relational data** can be represented

→ For regular data, the semi-structured representation is **highly redundant**

## Representation of irregular data

The structure is not fixed and allows missing values, duplicates, etc.

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

- Like for relational data, we can use **IDs** to refer to values elsewhere in the document



# XML documents

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

**Labeled** means that each node has a **label**

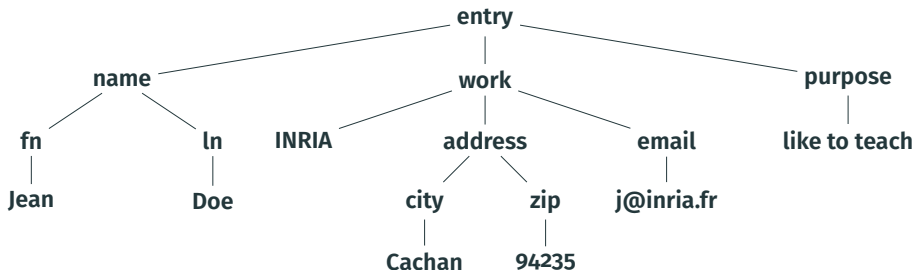
**Unranked** means that the number of children of a node is **unbounded**

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

XML specifies nothing more than a **syntax**: no meaning is attached to the labels.

A **dialect**, on the other hand, associates a meaning to labels (e.g., **title** in XHTML).

# XML documents are trees



## Remark

Some low-level software works on the serialized representation of XML documents, e.g., SAX (a parser and an API).

# Serialized representation of XML documents

Documents can be **serialized**, such as, for instance:

```
<entry><name><fn>Jean</fn><ln>Doe</ln></name><work>INRIA  
<email>j@inria.fr</email></work></entry>
```

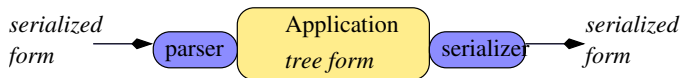
or with **pretty-printing** as:

```
<entry>  
  <name>  
    <fn>Jean</fn>  
    <ln>Doe</ln>  
  </name>  
  <work>  
    INRIA  
    <email>j@inria.fr</email>  
  </work>  
</entry>
```

Use `xmllint --format` and `pygmentize -l xml`

## Serialized form, tree form

Typically, an application gets a document in *serialized form*, parse it in *tree form*, and *serializes* it back at the end.



- The **serialized form** is the textual, linear representation of the tree
- Standardized **object-oriented model** for the tree form: the *Document Object Model* (DOM) which we saw last week. Also standardizes things relevant to Web browsers.

## XML basic syntax

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

```
<document/>
```

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

```
<document>
```

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

```
</document>
```

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

```
<document>
```

```
  <salutation color="blue">Hello World!</salutation>
```

```
</document>
```

# From serialized to tree form: text and elements

The basic components of an XML document are **element** and **text**.

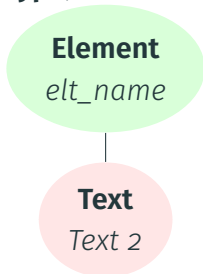
Here is an **element**, whose content is a **text**.

```
<elt_name>
```

```
  Textual content
```

```
</elt_name>
```

The tree form of the document, modeled in DOM: each node has a **type**, either **Document** or **Text**.



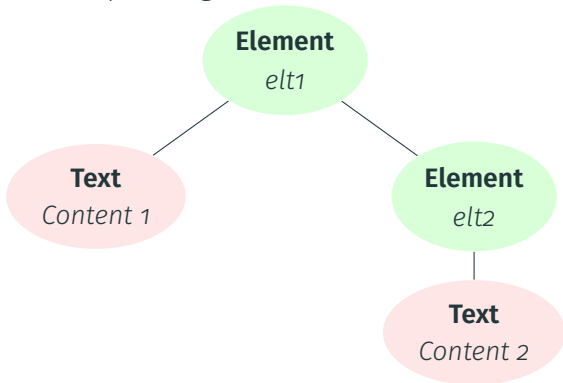
# From serialized to tree form: nesting elements

The **content** of an element is

1. the part between the opening and closing tags (in serialized form),
2. the subtree rooted at the corresponding **Element** node (in DOM).

Example of an element nested in another element.

```
<elt1>  
  Textual content  
  <elt2>  
    Another content  
  </elt2>  
</elt1>
```



## From serialized to tree form: attributes

**Attributes** are pairs of name/value attached to an element.

1. as part of the opening tag in the serialized form,
2. as special child nodes of the **Element** node (in DOM).

The content of an attribute is always **text** (no nesting).

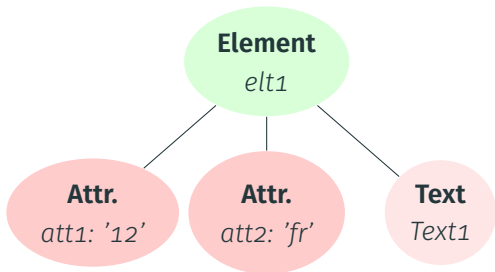
An element with two attributes.

```
<elt1 att1='12' att2='fr'>
```

Textual content

```
</elt1>
```

Attributes are *not* ordered, and no element can have two attributes with the same name.





## From serialized to tree form: the document root

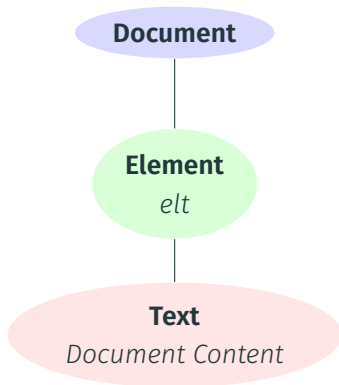
There may be a **prologue**, in which case it is the first document line:

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

The document must **always** be enclosed in one single **element root**  
(= a **tree**, not a **forest**)

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

Note: there may be other syntactic objects after the prologue  
(e.g., processing instructions).



# Summary: syntax and vocabulary

## Serialized form

- A document may begin with a **prologue**;
- It has a single **root element**;
- Each **opening tag** `<name>` has a corresponding **closing tag** `</name>`; everything between is either text or properly enclosed tag content.

## Tree form

- A document is a tree with a **root node** (**Document** node in DOM),
- The root node has exactly one element child (**Element** node), called the **element root**
- Each element node is the root of a **subtree**

## Entities and references

Entities are used for the physical organization of a document.  
An entity is **declared** (in the document type), then **referenced**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE a [  
  <!ENTITY myName "John Doe">  
  <!ENTITY mySignature SYSTEM "signature.xml">  
>
```

```
<a>  
  My name is &myName;.
```

```
  &mySignature;  
</a>
```

## Predefined entities

Five **predefined entities**, which in particular allow **escaping**:

Declaration	Reference	Symbol.
<code>&lt;!ENTITY lt "&lt;"&gt;</code>	<code>&amp;lt;</code>	<code>&lt;</code>
<code>&lt;!ENTITY gt "&gt;"&gt;</code>	<code>&amp;gt;</code>	<code>&gt;</code>
<code>&lt;!ENTITY amp "&amp;"&gt;</code>	<code>&amp;amp;</code>	<code>&amp;</code>
<code>&lt;!ENTITY apos "'"&gt;</code>	<code>&amp;apos;</code>	<code>'</code>
<code>&lt;!ENTITY quot """&gt;</code>	<code>&amp;quot;</code>	<code>"</code>

Also **numeric character references** based on **Unicode codepoints**:

- `&#42;` (decimal),
- `&#x2A;` (hexadecimal).

## Problem with entities: XML external entity attack

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

- On badly configured systems, this can exfiltrate `/etc/passwd` to an attacker

## Problem with entities: Billion laughs attack

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

→ This can **crash** a badly configured XML parser

## Comments and instructions

**Comments** can be put at any place in the serialized form.

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

They appear as **Comment** nodes in the DOM tree (they are typically ignored by applications).

**Processing instructions**: specific commands, useful for some applications, simply ignored by others.

The following instruction requires the transformation of the document by an XSLT stylesheet

```
<?xml-stylesheet href="prog.xslt" type="text/xslt"?>
```

## Literal sections

**Problem:** what if we do *not* want the content to be parsed?

```
<?xml version='1.0'?>
<program>
if ((i < 5) && (j > 6))
    printf("error");
</program>
```

**Solution:** use entities to escape, or use a **literal section**.

```
<?xml version='1.0'?>
<program>
<![CDATA[if ((i < 5) && (j > 6))
    printf("error");
]]>
</program>
```



## Interpreting labels: Namespaces

A particular label, e.g., *job*, may denote different notions in different contexts, e.g., a hiring agency or a scheduler.

The notion of `namespace` is used to distinguish them.

```
<doc xmlns:hire='https://a.hire.com/schema'  
      xmlns:sched='https://b.scheduler.com/schema' >  
  ...  
  <hire:job> ... </hire:job> ...  
  <sched:job> ... </sched:job> ...  
</doc>
```

# **XML dialects and standards**

---

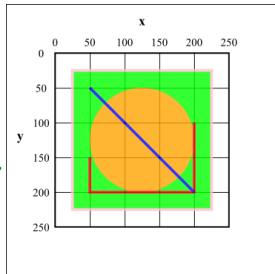
# Dialects

- XML is just an **abstract syntax** with no specific **meaning**
- An XML **dialect** specifies the **syntax** (which elements, attributes, etc.) and gives it a **semantics**
- Distinguish:
  - **Well-formedness**: an XML document is **well-formed** if it conforms to the XML spec (dialect-independent)
  - **Valid**: an XML document is correct relative to the syntax of a dialect (as specified, e.g., by a DTD, Schema, etc.)

# Popular XML dialects: SVG

An XML dialect for two-dimensional vector graphics:

```
<?xml version="1.0" encoding="UTF-8" ?>
<svg xmlns="http://www.w3.org/2000/svg"
  version="1.1">
  <rect x="25" y="25" width="200" height="200"
    fill="lime" stroke-width="4"
    stroke="pink" />
  <circle cx="125" cy="125" r="75" fill="orange" />
  <polyline points="50,150 50,200 200,200 200,100"
    stroke="red" stroke-width="4" fill="none" />
  <line x1="50" y1="50" x2="200" y2="200"
    stroke="blue" stroke-width="4" />
</svg>
```



## Popular XML dialects: DocBook

An XML dialect for technical documentation:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xml:id="simple_book"
  xmlns="http://docbook.org/ns/docbook" version="5.0">
  <title>Very simple book</title>
  <chapter xml:id="chapter_1">
    <title>Chapter 1</title>
    <para>Hello world!</para>
    <para>I hope that your day is proceeding
      <emphasis>splendidly</emphasis>!</para>
  </chapter>
  <chapter xml:id="chapter_2">
    <title>Chapter 2</title>
    <para>Hello again, world!</para>
  </chapter>
</book>
```

## Popular XML dialects: RSS

An XML dialect for **articles** to subscribe to websites (also **Atom**):

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  <title>RSS Title</title>
  <description>This is an example of an RSS feed</description>
  <link>http://www.example.com/main.html</link>
  <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
  <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
  <ttl>1800</ttl>

  <item>
    <title>Example entry</title>
    <description>Here is some description.</description>
    <link>http://www.example.com/blog/post/1</link>
    <guid isPermaLink="false">7bd204c6-1655-4c27-aeee-53f933c5395f</guid>
    <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
  </item>
</channel>
</rss>
```

## Popular XML dialects: OpenDocument and OpenXML

- XML-based **open standards** for office documents
- OpenXML is used by **LibreOffice** and **OpenXML** by **Microsoft Office**
- A LibreOffice file (e.g., **.odt**) is actually a **zip archive** of several files, including XML documents

## Popular XML dialects: MathML

An XML dialect to describe the **semantics** and **display** of mathematical equations. Example for  $ax^2 + bx + c$ :

```
<math>
  <apply>
    <plus/>
    <apply>
      <times/>
      <ci>a</ci>
      <apply>
        <power/> <ci>x</ci> <cn>2</cn>
      </apply>
    </apply>
    <apply> <times/> <ci>b</ci> <ci>x</ci> </apply>
    <ci>c</ci>
  </apply>
</math>
```



## Popular XML dialects: SSML

Speech Synthesis Markup Language: annotate text for text-to-speech

```
<!-- ?xml version="1.0"? -->
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <metadata>
    <dc:title xml:lang="en">Telephone Menu: Level 1</dc:title>
  </metadata>
  <p>
    <s xml:lang="en-US">
      <voice name="David" gender="male" age="25">
        For English, press <emphasis>one</emphasis>.
      </voice>
    </s>
    <s xml:lang="es-MX">
      <voice name="Miguel" gender="male" age="25">
        Para español, oprima el <emphasis>dos</emphasis>.
      </voice>
    </s>
  </p>
</speak>
```

## Many more dialects

See [https://en.wikipedia.org/wiki/List\\_of\\_XML\\_markup\\_languages](https://en.wikipedia.org/wiki/List_of_XML_markup_languages)

## Example of XML data dumps

- **OpenStreetMap**: As a **Protocolbuffer** dump or as **XML** in a custom OSM dialect
- **Wikimedia**: As a **Mediawiki dump** in XML, with schema <https://www.mediawiki.org/xml/export-0.10.xsd>
- **Stack Exchange**: As a dumb **XML encoding** of relational data
- **data.gouv.fr**: 1347 datasets are in **XML** (out of 37 147)
- **arXiv**: The arXiv dump provides metadata information in **XML**
- **DBLP**: The DBLP dump is in **XML**

# XML Ecosystem

---

# XML standards

**SAX** (Simple API for XML) gives an API for XML documents seen as a **sequence of tokens** (its serialization).

**DOM** (Document Object Model) gives an API for the **tree representation** of HTML and XML documents (independent from the programming language)

**Schema languages** Specify the structure of documents in a dialect (DTD, XML Schema, Relax-NG, etc.) (**more to come**)

**XPath** (XML Path Language) is a language for addressing **portions** of an XML document (**more to come**)

**XQuery** is a query language to extract information from collections of XML documents

**XSLT** (Extensible Stylesheet Language Transformations), to specify how to transform XML documents into other documents.

## Lesser-known XML standards

**XLink** Link to other XML documents

**XPointer** Refer to a **specific point** in an XML document

**XInclude** **Include** XML documents in other documents

**XProc** Language to define **pipelines** of XML content

**XSL-FO** Markup language for documents to be rendered as **paginated documents**

# Generic XML tools

- **API:** XML or SAX
- Parsers and type checkers, e.g., `xmlLint`
- GUI (Graphical User Interfaces)
- Editors
- XML diff
- Etc.

# Dirty tools to work with XML

- `xmlstarlet`: dirty way to process XML on the command line, uses `libxml`

```
curl -s 'https://dblp.uni-trier.de/db/conf/icdt/icdt2018.xml' |  
xmlstarlet sel -t -m '//inproceedings/author/text()' -c . -n
```

- Find elements with the XPath `//inproceedings/author` and copy their value followed by a newline
- Actually producing `XSLT` under the scenes

- `xml2` and `2xml`: Convert XML to a `text-based format` and back: useful to work with CLI utilities



# XML Research

---

# Research about XML

- XML has inspired much **theoretical research** in the database theory community
  - E.g., **test-of-time** award at the PODS conference in 2015 and 2016
- Natural **practical applications** for **theoretical research**
- For instance, **tree automata** (cf later) for validation, querying, etc.
  - See **survey** by Schwentick, *Automata for XML—A survey*, JCSS 2007<sup>1</sup>
- Study of **query languages** such as **XPath**
  - E.g., Benedikt and Koch, *XPath Leashed*, ACM Computing Surveys, 2009<sup>2</sup>
- Also other query languages, e.g., **tree pattern queries**
- A bit **out of fashion** in favor, e.g., of queries on **graph data**

---

<sup>1</sup><https://www.sciencedirect.com/science/article/pii/S0022000006001085>

<sup>2</sup><https://infoscience.epfl.ch/record/166852/files/25-leashed.pdf>

# Bibliography

Specifications from the World Wide Web Consortium, [w3.org](http://w3.org):

- Document Object Model. [w3.org/DOM](http://w3.org/DOM).
- Extensible Markup Language. [w3.org/XML](http://w3.org/XML).
- XML Schema. [w3.org/XML/Schema](http://w3.org/XML/Schema).
- XML Query (XQuery). [w3.org/XML/Query](http://w3.org/XML/Query).
- Extensible Stylesheet Language. [w3.org/Style/XSL](http://w3.org/Style/XSL).

Books:

- S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, P. Senellart. *Web Data Management*. Cambridge University Press, 2011.  
<http://webdam.inria.fr/Jorge/>
- S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan-Kaufman, 1999.

## Sources

- Slide 20: [https://en.wikipedia.org/wiki/XML\\_external\\_entity\\_attack](https://en.wikipedia.org/wiki/XML_external_entity_attack)
- Slide 21: [https://en.wikipedia.org/wiki/BillionLaughs\\_attack](https://en.wikipedia.org/wiki/BillionLaughs_attack)
- Slide 26: [https://en.wikipedia.org/wiki/File:SVG\\_example\\_markup\\_grid.svg](https://en.wikipedia.org/wiki/File:SVG_example_markup_grid.svg)
- Slide 27: <https://en.wikipedia.org/wiki/DocBook>
- Slide 28: <https://en.wikipedia.org/wiki/RSS>
- Slide 30: <https://en.wikipedia.org/wiki/MathML>
- Slide 31: [https://en.wikipedia.org/wiki/Speech\\_Synthesis\\_Markup\\_Language](https://en.wikipedia.org/wiki/Speech_Synthesis_Markup_Language)