XML Concepts

Prof. Andrea Omicini
Distributed Systems / Sistemi Distribuiti L-A
A.Y. 2007-2008
Alma Mater Studiorum–Università di Bologna a Cesena
Outline

Introducing XML
XML Fundamentals
Document Types Definitions (DTDs)
Namespaces
Internationalisation
XML & CSS
DOM & SAX
Introducing XML
What is XML?

- A W3C Standard
  - [http://www.w3.org/XML/](http://www.w3.org/XML/)

- A mark-up language for text documents
  - derived from SGML (Standard General Markup Language)
    - ISO 8879, [http://www.iso.ch/cate/d16387.html](http://www.iso.ch/cate/d16387.html)
  - eXtensible Markup Language

- A meta-markup language
  - to define markup languages
  - such as XHTML, XSLT, XML Schema...

- A formally-defined text-based language
  - verifiable for well-formedness and validity
  - usable across platform and technologies
What XML is not?

XML is not
- a programming language
- a network-transport protocol
- a document presentation language
- a database (manager)

It can be used (and it is actually) in all of those contexts, but it remains a markup language
Why Markup Languages?

**Markup**
- encoding embodied in the document, specifying document properties, as well as properties of information contained
  - for instance, formatting instructions
  - more generally, structural / semantic information
    - knowledge vs. data

**Marks / Markups**
- tag used to qualify / label text chunks
  - e.g., HTML tags

**XML example**

```xml
@student>
  <studentname>
    <name>Carlo</name>
    <surname>Nervo</surname>
  </studentname>
  <studentnumber>0000145678</studentnumber>
  <course>2036</course>
</student>
```
XML: X for eXtensibility

Basic idea of XML
- a simple meta-language for humans and automata
- to build electronic documents
- allowing users to define ad hoc markup languages

Then,
- XML is quite free, in general
- it can be “extended”
  - actually, specialised
  - to define more specific ad hoc markup languages

No predefined XML markups, as it happens instead in HTML
- they need to be defined
  - who does define them?
    - can we do this? how?
Hey, too many Languages already!!

Application domains are more and more numerous
- complex
- specific

Special / specialised languages as the engineer's tools
- to represent, denote & express behaviours and computations

Engineers working with computational / ICT systems will be called to use a number of different artificial languages, but also
- to know and understand computational models and paradigms
- to select languages and paradigms
- to define and build new languages

“Laurea Specialistica in Informatica”
- “Linguaggi e modelli computazionali”, “Ingegneria del SW”
XML: Applications

XML per se is “small” & simple

languages defined via XML are instead so many and complex

XML Applications

XML-defined markup languages

defined through a precise syntax

- DTD or XML Schema

they may be either standard or custom

Most standard XML applications are W3C

such as

- XSLT
- XML Schema
- XHTML
XML for Portable Data

- Cross-platform, long-term data format
- passing XML data through space and time
- along with Unicode and text-base standard format
- Text, text, text
  - both data and markup
  - all in the XML file
- XML document structure simple & clear
  - easy to parse
  - well-documented
- That is why XML is already everywhere
<xml version="1.0" encoding="utf-8">
<docroot>
  <head>
    <title>This is my document.</title>
  </head>
  <body>
    <p>A list of things I like.</p>
    <list>
      <item>weekends</item>
      <item>good beer</item>
      <item>midnight snacks</item>
      <item>ice cream
        <list>
          <item>chocolate</item>
          <item>cookie dough</item>
          <item>white russian</item>
        </list>
      </item>
      <item>shade trees</item>
    </list>
  </body>
</docroot>
How XML Looks like from a Browser

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```xml
<docroot>
  <head>
    <title>This is my document.</title>
  </head>
  <body>
    <p>A list of things I like.</p>
    <ul>
      <li>weekends</li>
      <li>good beer</li>
      <li>midnight snacks</li>
      <li>ice cream</li>
      <li>chocolate</li>
      <li>cookie dough</li>
      <li>white russian</li>
      <li>shade trees</li>
    </ul>
  </body>
</docroot>
```
How to Work with XML

XML is text
- so any text-editor is perfectly fine

A number of XML editors around
- but typically, general text editors with some programming / Web-oriented capabilities are good enough, and often even better

Visualisation is a different matter
- browsers do something
- but XML is not a presentation language, so...
- we need to understand
- what an XML document is
- how XML works
What is an XML Document?

It can be

- A text file
- A record in a database
- A run-time construction in memory

... In any case, it can be handled and transmitted by any system capable of dealing with text documents

```xml
<student>
  <studentname>
    <name>Carlo</name>
    <surname>Nervo</surname>
  </studentname>
  <studentnumber>
    0000145678
  </studentnumber>
  <course>2036</course>
</student>
```
How does XML Work?

Who handles XML documents?
- after it has been produced
- how/why?

XML parsers
- devising out the structure of the XML document
- verifying well-formedness and basic respect of XML syntax

XML validating parsers
- when applicable
  - there is either a DTD or a Schema
- checking validity

Examples
- web browsers, word processors, database servers, drawing programs, spreadsheets, programs in some language, etc.
Where is XML actually used?

Everywhere already.
Lot to be written, still...

SGML is where it comes from
- HTML was the first successful application of SGML
  - but had obvious limitations
  - too complex
    - more than 150 pages
    - never implemented fully
  - too complex for the Internet
- SGML “Lite” (1996, Bosak, Bray et al.)
- XML 1.0 (February 1998)

Then, a flow
- namespaces, XSL (then XSLT + XSL-FO), XHTML, CSS integration, XLink + XPointer, XML Schema, DOM, etc.
XML Fundamentals
A Simple XML Document

[player>
    Carlo Nervo
</player>
This is a complete XML document. It can be stored / recorded / built in the form of a number of different files or even in other forms:

- Carlonervo.xml, player.txt
- A record in a database
- A memory area built by a CGI, and then transmitted
- Sent by a Web server, with MIME type application/xml or text/xml
The document contains a single **element** of type `player`. Such an element is delimited by the **tag** `player` between **start tag** `<player>` and **end tag** `</player>`. In between the tags lays the element’s **content** Carlo Nervo. Tags are **markup**, the most common form of markup, but there are other kinds. **Content** is **character data** including the white space between Carlo & Nervo.
Tag Syntax

Very similar to HTML tags
- at least superficially
- `<tag>` for start tags, `</tag>` for end tags
- `<tag />` for empty tags
  - tags with no content, like `<br />` or `<hr />`

XML is case sensitive
- so, `<player>` can not be closed by end tag `</Player>`
- NOTE: thus, pay attention to non-case sensitive technologies when combined with XML
  - HTML, JavaScript & XHTML, ...
XML Trees: A Simple Example

```xml
<player>
    <name>Carlo</name>
    <surname>Nervo</surname>
    <team current="yes">Bologna</team>
    <team current="no">Mantova</team>
</player>
```
An XML Document has a tree-like structure

- one and only one **root**
  - root element or document element
- each node element can have one or more **child elements**
  - each element has at least one **parent**
  - child elements from the same parent are **siblings**
- leaves are either content or empty elements

Well-formedness stems from here

- `<em><b>Wrong</b> XML</em>` is not permitted
- nesting needs to be perfect, overlapping not allowed
<biography>
 <name><first_name>Carlo</first_name> <last_name>Nervo</last_name></name> was born somewhere and did nothing really meaningful before becoming a football player. After playing many years in minor teams, such as <football_team>Mantova</football_team>, he finally moved to <football_team>Bologna</football_team>, where he exploded to become one of the most respected leaders of the team, and also a member of the <football_team>Italian National Team</football_team>.

... 

</biography>

XML Documents for written narrative, such as articles, reports, blogs, books, novels
- elements with mixed content
- not easy for automated processing and exchange
Elements can be labelled by **attributes**

- attributes are specified in the start tag
  - and in the only tag of empty elements
- any number of attributes can be in principle associated to an element

An attribute is a name-value pair of the form `name="value"

- alternative forms use single quotes instead of double quotes and spaces before / after the "equals" (=) sign
- only one attribute with a given name allowed per element

Attributes do not change the tree structures of an XML document

- but they are qualifiers for the nodes and leaves of the tree
Using Elements or Attributes?

Attributes are for meta-data about the element, and content is information of the element. Maybe, but then it is not easy to clearly distinguish between the two.

Element-based structure is more flexible than attribute-based:
- Attributes provide for a flat data structure / elements can be nested as needed.
- Attributes are unique within an element / any number of elements of the same type can be used within an element.

Attributes are quite useful in narrative-based XML documents.

```xml
<player>
    <name>Carlo</name>
    <surname>Nervo</surname>
    <team current="yes" value="Bologna" />
    <team current="no" value="Mantova" />
</player>
```
XML Names are used and are the same for the names of elements, attributes and some other constructs to increase efficiency and abate complexity.

An XML name can include:
- any letter
  - latin or even non-latin, like ideographs
- any digit
- underscore, hyphen and period (_,-,.)
- a colon (:) is reserved to namespaces

An XML name may not include other punctuation signs, nor any sort of white spaces
- and can begin only with letters, ideographs or underscore
An XML Parser interprets the character sequences it is fed with, trying to devise out its tree-like structure. So, for instance, '<' always taken as the beginning of a tag. What if we need a '<' character in the document, as in a JavaScript code?

All characters are interpreted as character data to be parsed unless an escape character '&' is encountered. Character data to parse start again after char ';'

E.g., the content of the element `<superheroes>Batman & Robin</superheroes>` becomes the parsed character data Batman & Robin.
Entity References

&entityreference;

- an entity is something defined outside the normal "flow" of the XML document
- out of the XML tree
- used for constants, common values, external values, etc.
- through an entity reference

Users of any sort may define their own entities
- we'll see how soon, for instance through DTDs
## Pre-defined XML Entities

<table>
<thead>
<tr>
<th>Markup</th>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>less-than</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>greater-than</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
<td>ampersand</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>single quote</td>
</tr>
</tbody>
</table>
Including code chunks from any language with < or " can be tedious
we need to say the parser "do not parse this"
good for instance to include segments of XML code to show

**CDATA Section**

between `<![CDATA[ and ]]>`

can contain anything but its own delimiters

After parsing, no way to tell where a text came from, a CDATA section or not
Comments

Easy!

<!-- Comment -->

It cannot contain --, nor it can end with -->

Comments do not affect the document tree-structure

they can appear anywhere, even before the root element

but not inside a tag or a comment

Parsers may either drop or keep them at their will

Comments are meant to improve human legibility of XML docs

to give info to a computational agents, processing instructions
XML Processing Instructions

Need to pass information for a given application through the parser
comments may disappear at any stage of the process

Processing instructions have this very end

```xml
<?target ... ?>
```

The target may be the application that has to handle, or just an
identifier for the particular processing instruction

```xml
<?php ... ?>
<?xml-stylesheet ... ?>
```

A processing instruction is markup, not an element
it can appear everywhere out of a tag, even before or after the root
The XML Declaration

- Looks like an XML processing instruction
  - but it is not: just the XML declaration
- It is optional
  - but if there, should be the first thing in the document, absolutely
    - not even comments allowed before

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

- Version is the XML version (1.0, 1.1, ...)
- Encoding is the form of the text (Unicode in the example)
  - optional, default Unicode
- Standalone means that it has no external DTD
  - optional, default "no"
Checking Well-Formedness

Main rules
- perfect match between start and end tags
- no overlapping elements
- one and only one root elements
- attribute values are always quoted
- at most one attribute with a given name per element
- neither comments nor processing instructions within tags
- no unescaped > or & signs in the character data of elements or attributes

Tools on the Web
- Just look around
DTD
XML is flexible
  whatever this means
  but sometimes flexibility is not a feature within a given application scenario

Sometimes, some strict rule is required
  some control over syntax should be enforced
    like, a football player should have at least one team

**Document Type Definition** (DTD)
  to define which XML documents are valid

Validity is not mandatory as well-formedness
  how to handle errors is optional
A **valid XML Document** includes a DTD the document satisfies

**Main principle**
- everything not permitted is forbidden
  - that is, DTDs specifies *positive* examples

**Everything in the XML document must match a DTD declaration**
- then, the document is **valid**
- otherwise, the document is **invalid**

**Many things a DTD does not say**
- we stick with what we can specify
DTD is...

- SGML-based
  - syntax a bit awkward
  - but after all easy to understand
  - and quite suited for short and expressive descriptions

- It allows XML designers to define a grammar for their documents
  - typical syntax-based approach
  - maybe limited, but easy to implement

- Maybe, DTD is not the future of XML document validation
  - XML Schema should be that
  - but understanding DTDs, how to modify them, how to write your own ones, is likely to be useful or maybe necessary for a while, still
We do not go too deep into DTD syntax, we just look at the example above, and comment.
DTD Declaration

<?xml version="1.0" standalone="yes"?>
<!DOCTYPE football_player [
    <!ELEMENT player (name, surname, team+)>
    <!ELEMENT name (#PCDATA)> 
    <!ELEMENT surname (#PCDATA)> 
    <!ELEMENT team (#PCDATA)> 
    <!ATTLIST team current (yes | no) #REQUIRED> 
]
[player>
    <name>Carlo</name>
    <surname>Nervo</surname>
    <team current="yes">Bologna</team>
    <team current="no">Mantova</team>
</player>

DTD is declared here as internal
--- but could be declared separately
<!DOCTYPE football_player SYSTEM "football_player.dtd">
--- even referring to an external / shared resource
<!DOCTYPE football_player SYSTEM "http://..."
So, you may define your own DTD, and either include it in your XML document or save it as an independent document, and refer from one or more XML docs or use an external DTD defined by someone else like, a working group you belong to, or a standardisation body of any sort by referring to that externally-defined syntax for your XML docs.
A player element contains one name, one surname and one or more teams in that precise order and they are just parsed character data (#PCDATA).
Some Syntax

"","" is for sequence
to define ordered lists

" | " is for choice
to provide for alternatives

suffixes
"*" for zero or more occurrences
"+" for one or more occurrences
"?" for zero or one occurrence

parenthesis for grouping
at any level of indentation
operators and suffixes applicable to any level

ANY for free-form content
EMPTY for empty element
A team element has a current attribute which is mandatory. #IMPLIED would say optional, instead and can be either yes or no enumeration as an attribute type.
Attribute Defaults

- **#IMPLIED**: the attribute is optional
- **#REQUIRED**: the attribute is mandatory
- **#FIXED**: either it is explicitly specified or not, it has a given value
- "literal": the default value is the "literal" quoted string
Attribute Types

- **CDATA**: any string of text acceptable in a well-formed XML attribute value
- **NMTOKEN, NMTOKENS**: more than an XML name: anything accepted as the first character. The plural form accepts more than one separated by whitespaces.
- **ENTITY, ENTITIES**: name(s) of unparsed entities declared elsewhere in the document
- **ID**: an XML name unique in the document, working as an identifier
- **IDREF, IDREFS**: reference(s) to IDs in the documents
- **NOTATION**: name of a notation used & defined in the document (rare!!)
- **enumeration**:
Other DTD Declarations, etc.

ENTITY declarations
<!ENTITY footer SYSTEM "http://lia.deis.unibo.it/~ao/footer">

NOTATION declarations
  who cares actually

We stop here
  more only for those who need it
Namespaces
What are Namespaces for?

Distinguish
- Different XML applications may use the same names at any scale, from personal to world-wide
- A namespace allows them to be clearly distinguished

Group
- Names of elements and attributes of the same XML application can be grouped together to be more easily recognised and handled

Example: `<set>` is an element in both SVG and MathML applications
- What if I have to use them together?
- Namespaces can be used to disambiguate names
Syntax for Namespace Use

Qualified names
- prefix : local_part

Examples of qualified names
- or QNames, or raw names
- rdf:description, xlink:type, xsl:template

Used for both element and attribute names
Associating Prefixes to URI

Example

A large firm could have a number of namespaces for different purposes:

```xml
<company
    xmlns:local="http://www.company.it/xml/"
    xmlns:euro="http://www.company.eu/xml/"
    xmlns:world="http://www.company.com/xml/">

then, you can use local, euro and world everywhere as prefixes
typically declared in the topmost element, but could be declared anywhere

example: <rdf:RDF xmlns:rdf="http://www.w3c.org/TR/REC-rdf-syntax#">

URI are standardised, not prefixes
but usually svg, rdf and other prefixes are not re-defined
also, they are conventional names
not necessarily pointing to an actually resource
Setting Default Namespaces

- **xmlns attribute**
  - alone, no suffix
  ```xml
  <svg xmlns="http://www.w3c.org/2000/svg" width="..." height="...">...
  </svg>
  ```
  - all the elements inside (including svg) are implicitly associated to the namespace `http://www.w3c.org/2000/svg`
  - no need for the svg prefix made explicity
Internationalisation
What does Text Mean?

“Text” can be encoded according to many different alphabets, mapping between characters and integers (code points). Character set, ASCII being the most (un)famous, now Unicode. A character encoding determines how code points are mapped onto bytes. So, a character set can have multiple encodings. UTF-8 and UTF-16 are both Unicode encodings. Any XML document is a text document. So, encoding should be declared.
The XML Encoding

Part of the XML Declaration

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

Most common values

- utf-8, utf-16 *(Unicode)*
- ISO-8859-1 *(Latin-1)*

See also: XML-Defined Character Sets

- Unicode and ISO are the most used families

Used also for external parsed entities

- like DTD fragments, or XML chunks
- which may have different encodings
- there, version may be dropped

It is a text declaration, but no longer a XML declaration
Multi-Lingual Documents

Example: a spell-checker, or a voice-reader parsing an XML doc

How to determine the language of a subpart?

- for multi-lingual docs

xml:lang attribute
- can be associated to any element
- determines the language of the element

Values are to be found in ISO 639
- standard: two letters for each language known
- if not there, IANA
  - prefix i-
  - such as i-navajo, i-klingon, ...
- if not there, too, such as for user-defined tags
  - prefix x-
  - such as x-quenya
Encoding for Portability

Working around encoding is not simply an “internationalisation” issue; it is also about *portability*.

When transmitting / communicating through text-based files, many errors typically occur, which are often not easy to catch.

XML abilities to:
- handle encoding precisely and accurately
- embody encoding information within each document

make it a powerful tool for easy and hassle-free portability across platforms, across applications, across time.
XML on Browsers

Different experiences with different browsers when trying to visualise an XML document.

XML however can be transformed to become easier to handle by standard browsers.

Two main approaches:
- Web-based one: XML + CSS
- XML-based one: XSL

In the following we explore the XML + CSS issue.
Cascading Style Sheets (CSS)

- a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents

Standard W3C

- http://w3c.org/Style/CSS

Goals

- describing how to present elements of a document
  - spanning over a range of different media
  - separating style description from content and structure

In this course we assume that you already know the basics
- if not, look at http://www.w3.org/Style/CSS/learning
/* Work around a Mozilla bug */
POEM { display: block }

/* Make the title look like an H1 header */
TITLE { display: block; font-size: 16pt; font-weight: bold }
POET { display: block; margin-bottom: 10px }

/* Put a blank line in-between stanzas,
only a line break between verses */
STANZA { display: block; margin-bottom: 10px }
VERSE { display: block }
XML + CSS

Any XML documents can be prepared for browser visualisation via CSS
Two things needed
- a CSS style sheet referring to the proper elements types of the XML document
- the association between the XML document and the CSS style sheet

Processing directive
to associate CSS to XML

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

CSS style sheet defining presentation style for the XML document tags

```
nometag {
  attributo1 : valore1;
  ...
}
```

No need for DTD or Schema
even though the browser could anyway complain...
<?xml version="1.0"?>
<xml-stylesheet type="text/css" href="poem.css"/>
</POEM>

<TITLE>Darest Thou Now O Soul</TITLE>
<POET>Walt Whitman</POET>

<VERSE>Darest thou now 0 soul,
<VERSE>Walk out with me toward the unknown region,
<VERSE>Where neither ground is for the feet nor
<VERSE>any path to follow?</VERSE>

<VERSE>No map there, nor guide,
<VERSE>Nor voice sounding, nor touch of
<VERSE>human hand,
<VERSE>Nor face with blooming flesh, nor lips,
<VERSE>are in that land.</VERSE>

<VERSE>I know it not 0 soul,
<VERSE>Nor dost thou, all is blank before us,
<VERSE>All waits undream'd of in that region,
<VERSE>that inaccessible land.</VERSE>

<VERSE>Till when the ties loosen,
<VERSE>All but the ties eternal, Time and Space,
<VERSE>Nor darkness, gravitation, sense,
<VERSE>nor any bounds bounding us.</VERSE>

<VERSE>Then we burst forth, we float,
<VERSE>In Time and Space 0 soul,
<VERSE>prepared for them,
<VERSE>Equal, equip at last, (0 joy! 0 fruit of all)
<VERSE>them to fulfill 0 soul.</VERSE>
Example: How Mozilla Visualises it [without CSS Style Sheet]

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<TITLE>Darest Thou Now O Soul</TITLE>
<POET>Walt Whitman</POET>
<STANZA>
<VERSE>Darest thou now O soul</VERSE>
<VERSE>Walk out with me toward the unknown region</VERSE>
<VERSE>Where neither ground is for the feet nor any path to follow</VERSE>
</STANZA>
<STANZA>
<VERSE>No map there, nor guide</VERSE>
<VERSE>Nor voice sounding, nor touch of human hand</VERSE>
<VERSE>Nor face with blooming flesh, nor lips, are in that land</VERSE>
</STANZA>
<STANZA>
<VERSE>I know it not O soul</VERSE>
<VERSE>Not dost thou, all is blank before us</VERSE>
<VERSE>All waits upstream'd of in that region, that inaccessible land</VERSE>
</STANZA>
<STANZA>
<VERSE>Till when the ties looser</VERSE>
<VERSE>All but the ties eternal, Time and Space</VERSE>
<VERSE>Not darkness, gravitation, sense, nor any bounds bounding us</VERSE>
</STANZA>
<STANZA>
<VERSE>Then we burst forth, we float</VERSE>
<VERSE>In Time and Space O soul, prepared for them</VERSE>
<VERSE>Equal, eq uit at last, (O joy! O fruit of all!) them to fulfill O soul</VERSE>
</STANZA>
```
Darest Thou Now O Soul
Walt Whitman

Darest thou now O soul,
Walk out with me toward the unknown region,
Where neither ground is for the feet nor any path to follow?

No map there, nor guide,
Nor voice sounding, nor touch of human hand,
Nor face with blooming flesh, nor lips, are in that land.

I know it not O soul,
Nor dost thou, all is blank before us,
All waits undream'd of in that region, that inaccessible land.

Till when the ties loosen,
All but the ties eternal, Time and Space,
Nor darkness, gravitation, sense, nor any bounds bounding us.

Then we burst forth, we float,
In Time and Space O soul, prepared for them,
Equal, equipt at last, (O joy! O fruit of all!) them to fulfil O soul.
DOM & SAX
Manipulating XML

Representing information in an XML Document and presenting it somehow is not enough for most non-trivial application scenarios. Mostly, we often need to manipulate access, delete, modify parts of an XML document which either may or may not be and XML file. This is typically done through programming language of many sorts through ad hoc API. The most used / hated / deprecated / widespread are DOM and SAX.
Document Object Model

http://www.w3.org/DOM/
standard W3C, as usual

"The **Document Object Model** is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents"

It applies to HTML as well as XML
It is essentially an API
standardised for Java & ECMAScript
but can be extended to other languages

There is no time here to go deep into DOM
we just try to understand its nature, goals and scope
DOM & Levels

DOM views an XML tree as a data structure similar to the DOM from Javascript.

DOM loads the whole XML document in memory to manipulate it, maybe huge memory consumption.

It is quite large and complex...

- Level 1 Core: W3C Recommendation, October 1998
  - primitive navigation and manipulation of XML trees
  - other Level 1 parts: HTML

- Level 2 Core: W3C Recommendation, November 2000
  - adds Namespace support and minor new features
  - other Level 2 parts: Events, Views, Style, Traversal and Range

- Level 3 Core: W3C Working Draft, April 2002
  - adds minor new features
An XML document is a tree. The tree contains nodes, one of them is a root node. Nodes possibly have siblings, children, one parent, content, tag, etc.

The DOM specification states that a node can contain document, doc. fragment, doc. type, element, attribute, processing instruction, comment, text, CDATA section, entity, notation.

It also defines which kind of child nodes they should / could have.
Properties & Methods of DOM

Every DOM node has **properties** and **methods** to explore and update the XML tree.

Every DOM node has a **name**, a **value**, a **type**

There are general properties and methods for all kinds of nodes:
- **attributes** returns all the attributes of the node
- **appendChild(newChild)** appends `newChild` after the other child nodes

Then, any specific kind of node has its own specific properties and methods:

These properties and methods are made available by the suitable API for the language of choice:
- many solutions for Java
public static void main(String[] args) {
    try {
        DOMParser p = new DOMParser();
        p.parse(args[0]);
        Document doc = p.getDocument();
        Node n = doc.getDocumentElement().getFirstChild();
        while (n != null && !n.getNodeName().equals("recipe"))
            n = n.getNextSibling();
        PrintStream out = System.out;
        out.println("<?xml version="1.0"?>");
        out.println("<collection>");
        if (n != null)
            print(n, out);
        out.println("</collection>");
    } catch (Exception e) {e.printStackTrace();}
}
Main Problem of DOM

The XML document is loaded as a whole and handled altogether in memory:
- it might be time-consuming and difficult to manage
- wouldn't it be better if we could load only the part we are actually manipulating

This is the motivation behind SAX:
- which is not started as a standard
- has problems of acceptance
- but has indeed a long tail of followers
- and also its good reasons to exist
Differently from DOM, SAX is event-based. It sees the document not as a tree, but as a text document flowing through the SAX parser and generating events as soon as document started / ended, elements started / ended, character content, etc.

A very simple model
- good for simple applications
- and also to avoid memory abuse

Not so well-supported as DOM is
- in terms of standardisation
- as well as of tools