XML Basics
Extensible Markup Language (XML)
Technologie de l'e-commerce - Partim Théorie 1

Ludovic Kuty <ludovic.kuty@hepl.be>

Haute Ecole de la Province de Liège

2011 - 2012

Created on 19/09/2011 at 08:01

Outline I

Outline II

versions

- XML is a W3C recommendation
- Version 1.0
  - 5th edition
  - since 2008/11/26
  - http://www.w3.org/TR/xml/
- Version 1.1
  - 2nd edition
  - since 2006/09/29
  - http://www.w3.org/TR/xml11/

definition

XML = eXtensible Markup Language

- Meta-language with minimal syntax ⇒ used to define XML applications or vocabularies
- Gives meaning to data with tags
- Well defined basic syntax but no restriction on tags and structure

meaning with tags

- For very simple kind of documents, tags alone are enough to understand things
- But usually we need more informations to understand / process the document:
  - Rules to constrain data: a schema
  - Documentation to understand the meaning of data: books, articles, …
- Think about a complex MathML or SVG document
- How could you possibly understand what is going on without any visual clue ?

meaning with tags

examples

MathML

```
<mrow> <mrow> <mrow> <mo>(</mo> <mrow> <mrow> <mn>2</mn> <
mo>&#8290;</mo> <mi>k</mi> </mrow> <mo>-</mo> <mn>1</mn> </
mrow> <mo>)</mo> </mrow> <mo>!!</mo> </mrow> <mo>&#8290;</
mo> <mrow> <munderover> <mo>&#8721;</mo> <mrow> <msub> <mi>
i</mi> <mn>1</mn> </msub> <mo>=</mo> <mn>1</mn> </mrow> <mi
>n</mi> </munderover>
```

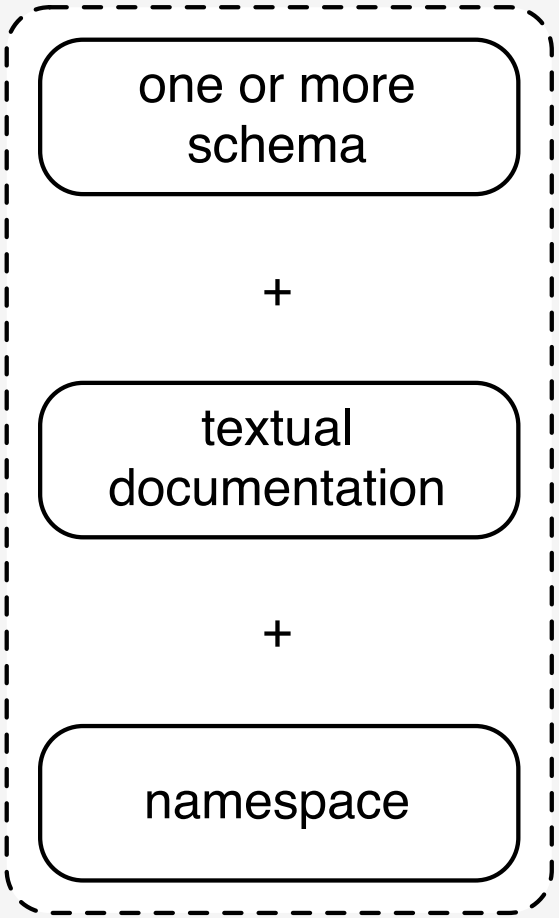
meaning with tags

examples

SVG

```
<path d="M136.128 28.837C129.728 29.637 104.999 5.605
119.328 37.637C136.128 75.193 60.394 76.482 44.128 65.637
C27.328 54.437 51.328 84.037 51.328 84.037C69.728 104.037
35.328 87.237 35.328 87.237C0.928 74.437 -23.072 100.037
-26.272 100.837C-29.472 101.637 -34.272 104.837 -35.072
98.437C-35.872 92.037 -42.989 75.839 -75.073 101.637C
-98.782 120.474 -111.655 100.11 -111.655 100.11L-115.655
107.237C-137.455 70.437 -124.236 128.765 -124.236 128.765C
-115.436 163.165 16.128 121.637 16.128 121.637C16.128
121.637 184.928 91.237 196.129 87.237C207.329 83.237
299.911 89.419 299.911 89.419L294.529 71.092C229.729 24.691
 212.929 49.637 199.329z"/>
```

vocabularies



meaning with tags

relational comparison

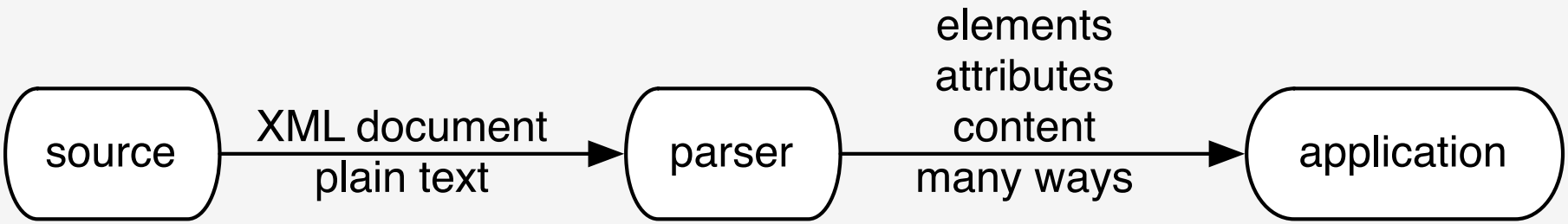
Like in the database world:

- A relational schema gives structure to data using attributes (name and type). Think about the columns of a table. Just like an XML schema
- A relation is data at any given point in time just like an XML document
- The data conforms to the schema

power

- Given that freedom, XML is powerful to describe a lot of things
- But it is quite useless without partner technologies like schemas, namespaces, XPath, XQuery, XSLT, XSL-FO, XML Pipeline, XInclude, XPointer, XLink, XForms, …
- You can get them on the W3C Web site at
  `http://www.w3.org/standards/xml/`
- Not everything related to XML is made by the W3C. There are more elsewhere like Relax NG, Schematron, …

data flow



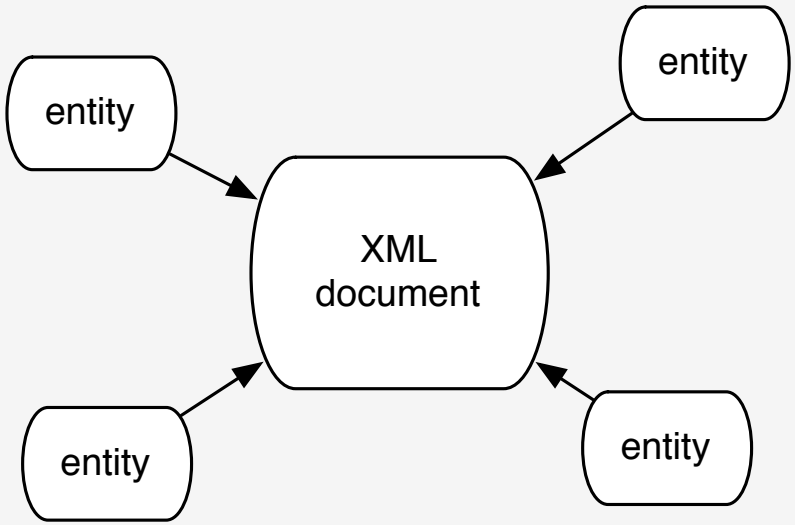
- The **source** provides an XML document. It can be a file, a database, the network, …
- The **parser** reads the document to extract informations (structure and content)
- The software **application** gets the data in a useable form

structure

- An XML document has a **physical** and a **logical** structure
- The final document is physically assembled with entities. Think about C and `#include` directives
- The logical document is composed of various objects wherever they come from. Think about a C program once the preprocessor has processed all `#include` directives. The directives have disappeared

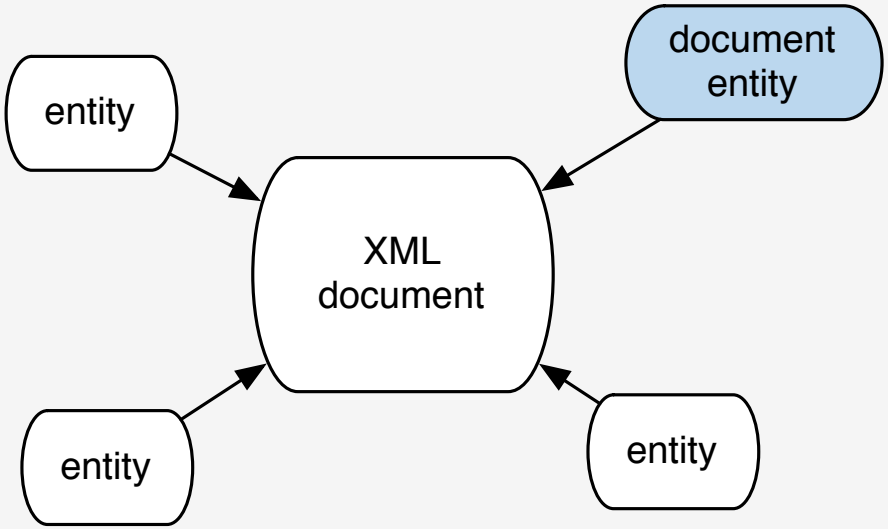
physical structure

- An XML document may consist of one or many storage units called **entities**
- They have content and are identified by their entity name (except for the document entity and the external DTD subset)



document entity

- The **document entity** serves as the root of the entity tree and a starting-point for a parser
- It has no name
- It might well appear on a processor input stream without any identification at all



entity

- An **entity** is a storage unit that is a piece of the XML document (or of the DTD)
- Has a name and a content called the **replacement text**
- Can be anything: a string, a file, a record in a DB, …
- Accessed through an **entity reference**

entity

- Entities come in two flavors:
  - **General entities** for **use** within the document content. Must conform to the XML rules
  - **Parameter entities** for **use** within the DTD. We'll see them later
- All entities are **declared** within the DTD
- Please note the difference between utilization and declaration.

general entities

Four types of general entities:

- Built-in or predefined entities
- Internal text entities. We'll see them later
- External text entities. We'll see them later
- Unparsed (binary) entities. We won't see them

general entity reference

- Points to an entity. It is like an alias or a pointer
- Number 6 in our example

Syntax

```
&name_of_entity;
```

built-in entities

Entity reference	Replacement text
`<`	<
`>`	>
`&`	&
`&quot;`	"
`&apos;`	'

- Used to escape special characters
- These characters can confuse the parser if written as is

logical structure

- Once the document has been assembled, it consists of various things or objects
- It represents the information stored in the document. This is what the application program gets[1]
- The things in question have been defined formally in a W3C recommendation called the **Infoset**
- XML Information Set
  - 2nd edition
  - since 2004/02/04
  - http://www.w3.org/TR/xml-infoset/

[1] More or less, since other data models are used by various APIs. Anyway, it is close to the Infoset so we get the right idea.

Infoset

- An XML document Infoset consists of a number of **information items**: document, element, attribute, processing instruction, unexpanded entity reference, character, comment, document type declaration, unparsed entity, notation and namespace
- Each information item (II) has a set of associated named **properties**
- We can see this as a tree structure but it can be made available to an application as anything else. E.g. event-based interface like Sax or TrAX

XML constituents

- What lies in an XML document ?
  - The XML declaration
  - Processing instructions
  - Elements
  - Attributes
  - Character references
  - Entity references
  - CDATA sections
  - Comments
  - Character data
- We are going to explain them all

first document

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE book SYSTEM "../DTD/book.dtd">
3  <?xml-stylesheet href="..." type="..."?>
4  <book added_at="2011/07/15 15:02">
5    <isbn>0-596-00197-5</isbn>
6    <title>Java &amp;amp; XML</title>
7    <author>
8      <firstname>Brett</firstname>
9      <lastname>McLaughlin</lastname>
10   </author>
11   <publisher>O'Reilly</publisher>
12   <edition>2nd</edition>
13 </book>
```

document structure

- An XML document contains only text
- It is divided in two parts:
  - The header contains all the declarations (XML, DOCTYPE and PI's)
  - The body contains XML datas within elements and attributes
- Comments and PI's may appear anywhere after the XML declaration

root element

- One element contains all the other elements
- It is called the **document element** or **root element**
- It is `book` in our example

XML declaration

- Optional but highly desirable
- First line in the document. Nothing before, not even a space !
- Gives information to the parser via 3 attributes: `version`, `encoding` and `standalone`
- Number 1 in our example

XML declaration

version attribute

- Matches the version of the XML specification used in the document
- Could be 1.0 or 1.1
- XML 1.1 is too complex for no real benefit. Thus always use 1.0

XML declaration

encoding attribute

- Character encoding used to write the document
- By default: UTF-8 (a superset of ASCII)
- If no indication, the parser may try to guess the encoding by using the first several bytes

XML declaration

standalone attribute

- Can be `yes` or `no` (by default)
- Indicates if the document makes use of things defined externally[2]
- Behavior depends on validating or not and 4 other factors. Useless complexity
- It is always ok to say `no` and it is the default
- Thus, *forget it* unless you want to check the details

[2] Indicates wether the external DTD subset affects the content of the document or not

characters basics

- A **character set** represents the characters available. Each character is associated with a number, its code point
- A **character encoding** specifies how to map code points to octets
- Famous encodings: ISO-8859-1 and UTF-8 for Unicode.

processing instructions

- Syntactically: `<?target instructions ?>`
- Given as is by the parser to the application ⇒ used to convey informations to the application
- PI's beginning with `xml` are forbidden
- Number 3 in our example

elements

- An **element** consists of:
  - A **start-tag** which name is an XML name like "book" or "title" in our example
  - A **content** = elements and/or character data. May be empty
  - An **end-tag**

The element `author` in our book example

- Element: lines 7 to 10 (with WSes)
- Start-tag: line 7 (without any WS)
- End-tag: line 10 (without any WS)
- Content: everything in between start-tag and end-tag. WSes after the start-tag on line 7, line 8 and 9, WSes before the end-tag on line 10

empty element

- An element may be empty
- It is written `<name></name>` without a single character (even a whitespace) between tags
- Shortcut, less error-prone: `<name/>`
- Note that it is allowed to contain attributes. Thus even if the content is empty, information may be conveyed in atrributes

XML name

- Contain alphanumeric characters from the character set, underscore _, dash - and dot .
- A letter or an underscore may begin a name
- Case sensitive
- The colon : is reserved for namespaces
- `http://www.w3.org/TR/xml/#NT-Name` in the specification

attributes

- Pairs a name and a value
- The attributes of an element are enclosed in the start-tag
- Order is not significant

Syntax

`name="value"` or `name='value'`

An attribute in our book example

In the document content, we have only one attribute named `added_at`.

elements vs attributes

- When to use elements and when to use attributes ?
- There is no definitive answer but we can give advice
- See [Effective XML] item 12

Attributes are...

- Good to store metadata, not the data itself. Think about `class` or `id` in XHTML
- Attribute values are just strings without structure (accessible to the parser)

Elements are...

- Good to store structured data
- And everything else that is not metadata

character references

- Refers to a specific character in the ISO/IEC 10646 character set
- Also called the Universal Character Set (UCS)
- Close to Unicode
- The character code is given in decimal or hexadecimal form

Decimal syntax

```
&#decimal;
```

Hexadecimal syntax

```
&#xhexadecimal;
```

character references

example

```
<?xml version="1.0" encoding="utf-8"?>
<text>&#x0BB9;&#x06B5;</text>
```

- Those characters have the glyphs 
- Check http://www.utf8-chartable.de/ for a table of Unicode characters with their UTF-8 encoding

CDATA sections

```
<![CDATA[
<p>Un extrait de code <i>HTML</i>
dans un document <b>XML</b></p>
]]>
```

instead of

```
&lt;p&gt;Un extrait de code &lt;i&gt;HTML&lt;/i&gt;
dans un document &lt;b&gt;XML&lt;/b&gt;&lt;/p&gt;
```

- Allows us to include raw text that won't be parsed as markup
- Frees us from using a lot of entity references

comments

```
<!--
Ceci est un
commentaire sur plusieurs
lignes.
-->
```

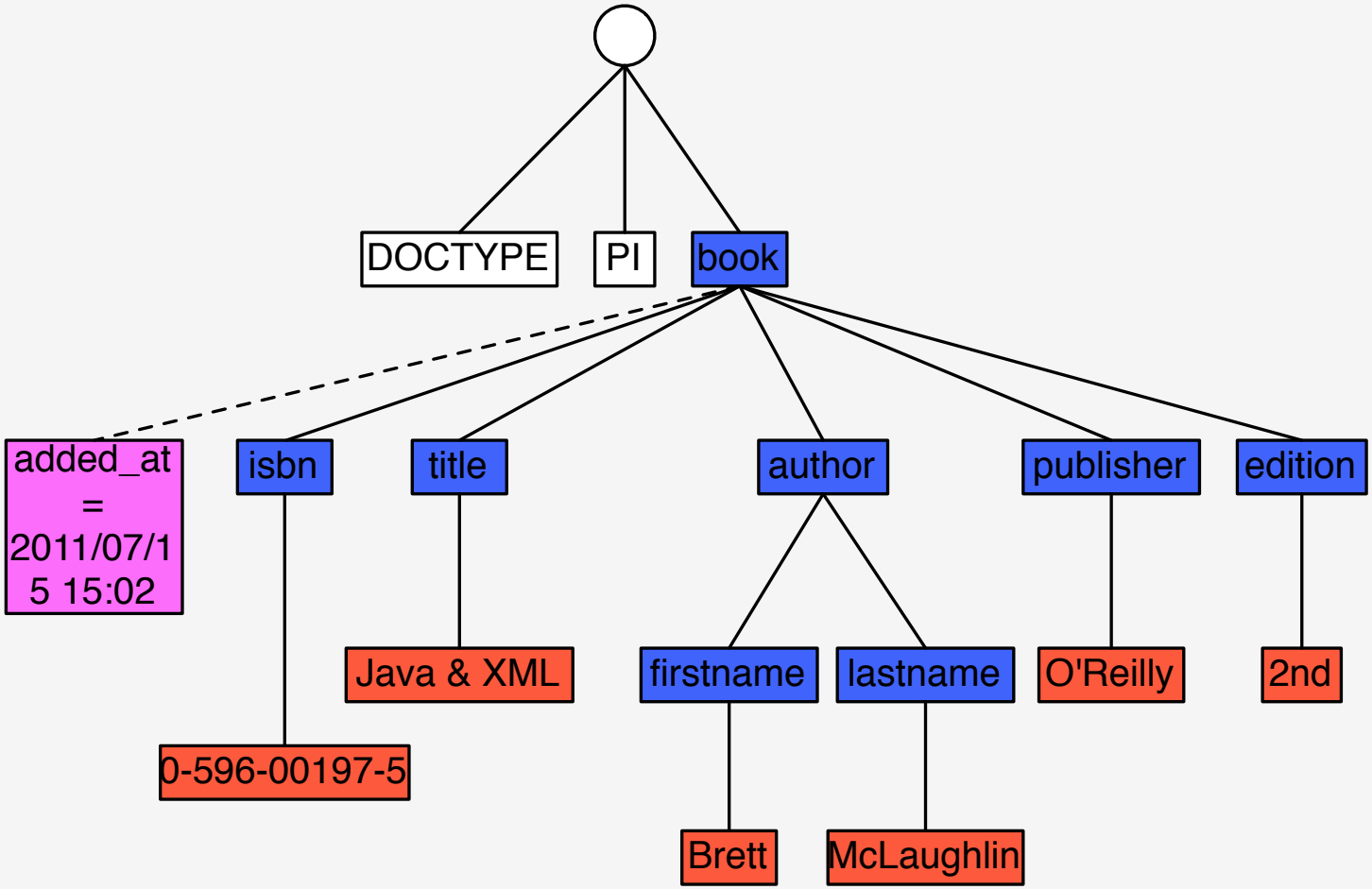
- Starts with a `<!--`
- Ends with a `-->`
- Characters `--` are forbidden inside comments

tree structure

- An XML document can be represented as a tree
- Useful with DOM, XSLT, XPath, XQuery, …
- The root of the tree does not map to anything in the serialized XML document. In particular, it is not the root element. In fact we can say that it is the document itself

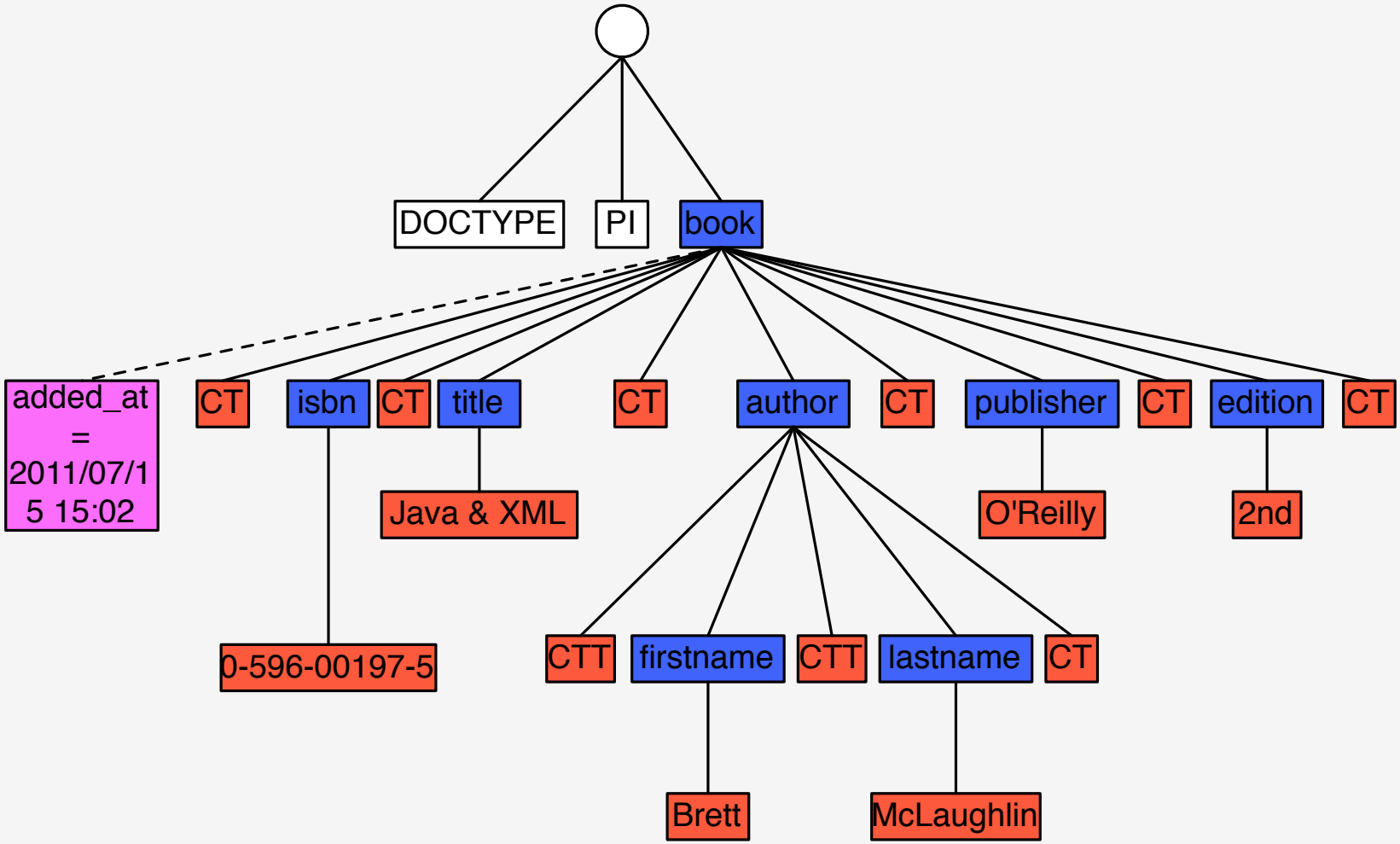
tree structure

without ignorable WS



tree structure

with ignorable WS



tree navigation

- We use the genealogical vocabulary to name things
- `title` is a **child** of `book` which is its parent
- `title` and `authors` are **siblings**
- `book` is the **ancestor** of every element node
- Every element node is a **descendant** of `book`

tree structure and data models

- Note that the tree structure presented here is quite loose. It is not a data model
- In fact, the only tool we have for now is the Infoset and it is quite abstract. It allows us to put names on the things in an XML document
- That means that every subsequent technology will have to define its own data model or reuse an existing one provided by another technology

well-formedness constraint

- If the document conforms to some basic rules, we say that it is **well-formed** because it respects the well-formedness constraint
- It is **mandatory** for every XML document. In fact a document can only be qualified as XML if it is well-formed
- A parser is **not allowed** to read a document that is not well-formed. Thus the parser has to stop parsing when it discovers a problem

some basic rules

- Every start-tag must have a matching end-tag
- Elements may nest but not overlap. We say that they must be properly nested
- There must be exactly one root element
- Attribute values must be quoted by double or single quotes
- No two attributes with the same name in the same element
- No unescaped < or & may occur in the character data of an element or an attribute

checking for well-formedness

ressources

- A graphical tool like Oxygen[3], Netbeans, Altova XMLSpy[4] or Stylus Studio[5]
- A Web site providing that kind of service like RUWF?[6],RXP[7] or STG[8]
- A parser and its API like Apache Xerces Java 2, JDOM, XOM, …We'll see some of them later

[3] http://www.oxygenxml.com/
[4] http://www.altova.com/
[5] http://www.stylusstudio.com/
[6] http://www.xml.com/pub/a/tools/ruwf/check.html
[7] http://www.cogsci.ed.ac.uk/~richard/xml-check.html
[8] http://www.stg.brown.edu/service/xmlvalid/

checking for well-formedness

sax.Counter in Apache Xerces Java 2

Java command

```
$ java -cp /Users/ludo/Library/Java/xerces-2_11_0/
xercesSamples.jar:/Users/ludo/Library/Java/xerces-2_11_0/
xercesImpl.jar sax.Counter examples/xml/book.xml
```

checking for well-formedness

sax.Counter in Apache Xerces Java 2

Well-formed

```
examples/xml/book.xml: 46 ms (7 elems, 1 attrs, 17 spaces,
36 chars)
```

Not well-formed

```
[Fatal Error] book.xml:5:30: The end-tag for element type "
title" must end with a '>' delimiter.
```

in french

English	French
entity	entité
entity reference	appel d'entité
character reference	appel de caractère
element	élément
tag	balise
start-tag	balise ouvrante
end-tag	balise fermante
well-formed	bien formé
valid	valide
well-formedness constraint	contrainte de forme
validity constraint	contrainte de validité