XML

Rodrigo García Carmona

Universidad San Pablo-CEU

Escuela Politécnica Superior

XML INTRODUCTION

THE XML LANGUAGE

- **XML:** Extensible Markup Language
  - Standard for the presentation and transmission of information.
  - An XML file is a text organized using tags.
  - Similar to HTML
    - But here tags structure content…
    - **…not the way it's displayed.**
  - Tree shaped.
  - Specially apt for streaming.
- This unit contains some samples and supplementary material. These can be found also as separate files so they can be played with. The slide will make a reference to them when appropriate.

XML RULES

- **Tag:** defines a piece of content. There are opening and a closing tag.
  - Begin with “<“ and ends with “>”. Closing tags begin with “</”.
- **Block:** content enclosed by a tag.
  - **Example:** `<title>My title</title>`
- **Attribute:** extra information provided by a tag. Inside the opening tag.
  - Surrounded by double quotes (“”). Attributes must have a value.
  - **Example:** `<book price="100">Harry Potter</book>`
- **The closing tag is optional:** only the opening tag.
  - Opening tag ends with “/>”. Have no block inside.
  - **Example:** `<remark text="Good" />`
- **Comments:** Not processed. Between “<!--" and “-->”.
- **XML is case-sensitive.** Spaces after the first are ignored.

XML EXAMPLE

```
<Bookstore> <!-- This is a bookstore -->
   <Book ISBN="ISBN-0-13-713526-2" Price="85" Edition="3rd">
      <Title>A First Course in Database Systems</Title>
      <Authors>
         <Author>
            <First_Name>Jeffrey</First_Name>
            <Last_Name>Ullman</Last_Name>
         </Author>
         <Author>
            <First_Name>Jennifer</First_Name>
            <Last_Name>Widom</Last_Name>
         </Author>
      </Authors>
      <Import />
   </Book>
   <Book ISBN="ISBN-0-13-815504-6" Price="100">
      <Title>Database Systems: The Complete Book</Title>
      <Remark>Buy this book bundled with "A First Course"!</Remark>
         <Authors><Author>
            <First_Name>Hector</First_Name>
            <Last_Name>Garcia</Last_Name>
         </Author>
      </Authors>
   </Book>
</Bookstore>
```

RELATIONAL MODEL VS. XML

Relational model

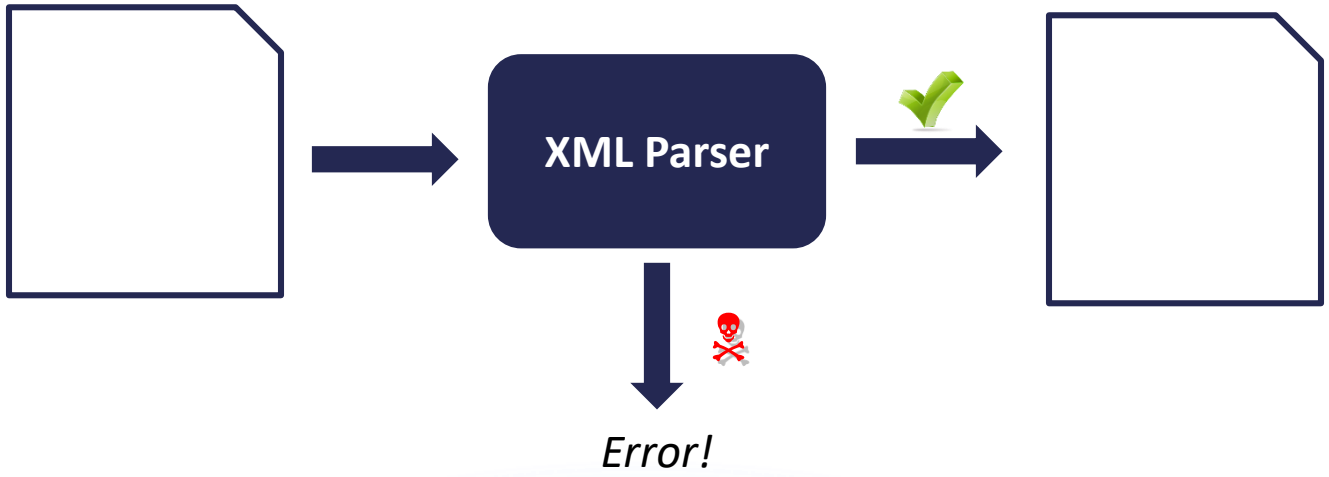
- Structure:
  - Tables
- Schema:
  - Predetermined
- Queries:
  - Easy and intuitive
- Ordering:
  - None
- Best for:
  - Machines, storage

XML

- Structure:
  - Tree hierarchy
- Schema:
  - Flexible, self-descriptive
- Queries:
  - A little more complex…
- Ordering:
  - Implicit
- Best for:
  - Humans, streaming

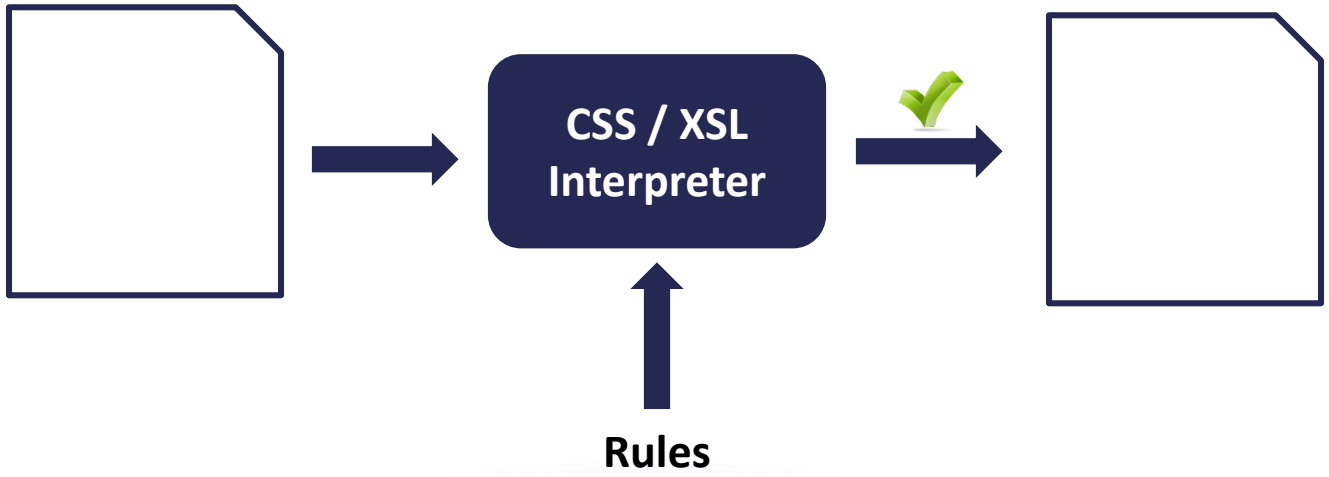
WELL-FORMED XML

- An XML document is **well-formed** if it complies with some **basic structural requirements**:
  - Only one root element.
  - Opening and closing tags are matched. Proper nesting.
  - Uniquely-names attributes in each element.
- **Parsers:** DOM, SAX…



SHOWING XML

- Rule-based languages can be used to transform XML into HTML:
  - **CSS:** Cascading Style Sheets.
  - **XSL:** eXtensible Stylesheet Language.
- The XML is processed by the parser first.



XML STANDARDS

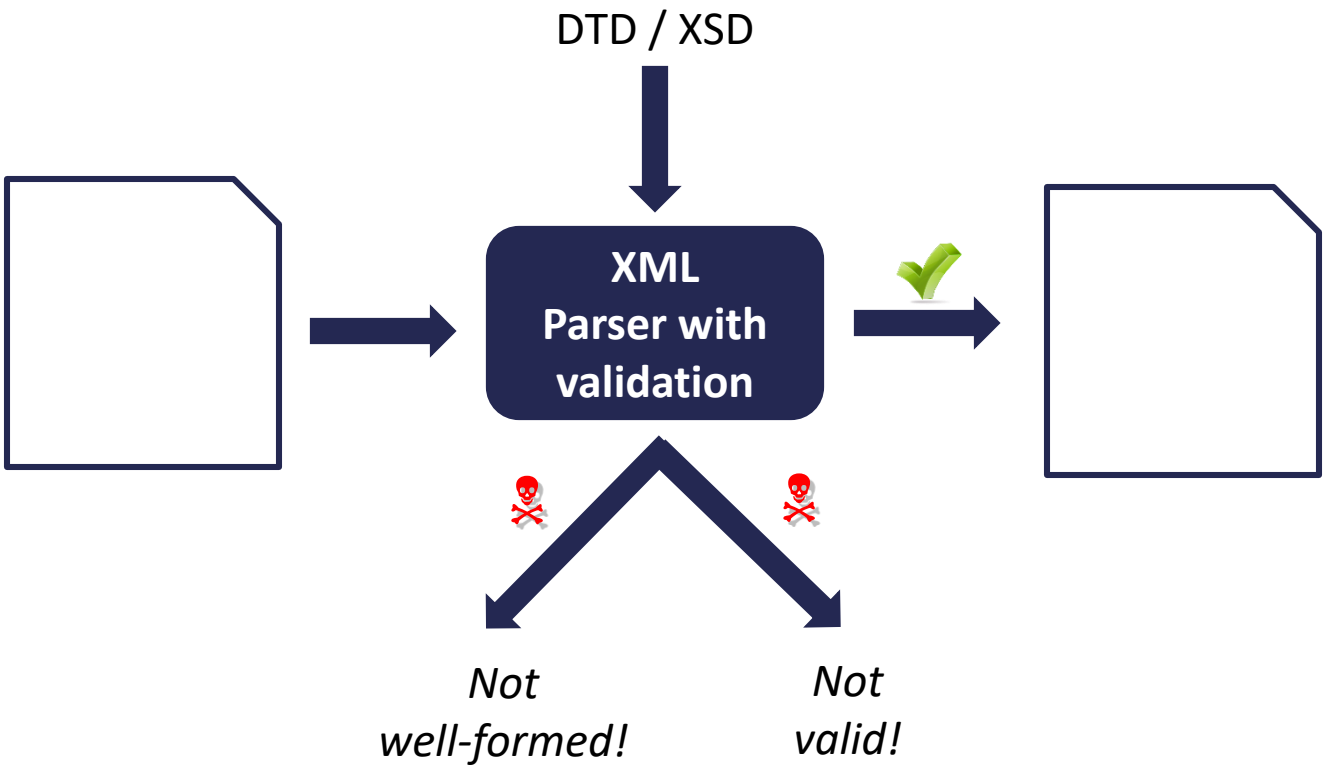
- XML is the most popular data representation and exchange format.
- There are plenty of standards that work alongside XML.
- These are the most important:
  - DTD
  - XSD
  - XPath
  - XQuery
  - XSL

DTD AND XML SCHEMA

VALID XML

- An XML document is **well-formed** if it complies with some **basic structural requirements**:
  - Only one root element.
  - Opening and closing tags are matched. Proper nesting.
  - Uniquely-named attributes in each element.
- An XML document is **valid** if it complies with some **content-specific requirements**. These requirements can be specified using two standards:
  - **DTD:** Document Type Descriptor.
  - **XSD:** XML Schema.
- This section uses the supplementary material marked as “01-DTD XSD”.

VALIDATING XML



DTD

- Document Type Descriptor.
- Simple standard to validate XML.
- **Provides a grammar that can be used to define:**
  - Elements
  - Attributes
  - Nesting
  - Ordering
  - Number of occurrences
- DTD has some special attributes (**untyped** pointers):
  - ID
  - IDREF / IDREFS
- The easiest way to learn DTD is by example!

SAMPLE DTD

```
<!DOCTYPE Bookstore [
   <!ELEMENT Bookstore (Book | Magazine)*>
   <!ELEMENT Book (Title, Authors, Remark?)>
   <!ATTLIST Book ISBN CDATA #REQUIRED
                  Price CDATA #REQUIRED
                  Edition CDATA #IMPLIED>
   <!ELEMENT Magazine (Title)>
   <!ATTLIST Magazine Month CDATA #REQUIRED Year CDATA #REQUIRED>
   <!ELEMENT Title (#PCDATA)>
   <!ELEMENT Authors (Author+)>
   <!ELEMENT Remark (#PCDATA)>
   <!ELEMENT Author (First_Name, Last_Name)>
   <!ELEMENT First_Name (#PCDATA)>
   <!ELEMENT Last_Name (#PCDATA)>
]>
```

XML FOR SAMPLE DTD

```
<Bookstore>
   <Book ISBN="ISBN-0-13-713526-2" Price="
100" Edition="3rd">
      <Title>A First Course in Database
Systems</Title>
      <Authors>
         <Author>
            <First_Name>Jeffrey</First_Name>
            <Last_Name>Ullman</Last_Name>
         </Author>
         <Author>
            <First_Name>Jennifer</First_Name>
            <Last_Name>Widom</Last_Name>
         </Author>
      </Authors>
   </Book>
```

```
<Book ISBN="ISBN-0-13-815504-6" Price="100">
      <Title>Database Systems: The Complete
Book</Title>
      <Authors>
         <Author>
            <First_Name>Hector</First_Name>
            <Last_Name>Garcia-Molina</Last_Name>
         </Author>
         <Author>
            <First_Name>Jeffrey</First_Name>
            <Last_Name>Ullman</Last_Name>
         </Author>
         <Author>
            <First_Name>Jennifer</First_Name>
            <Last_Name>Widom</Last_Name>
         </Author>
      </Authors>
      <Remark>
         Buy this book bundled with "A First
Course" - a great deal!
      </Remark>
   </Book>
</Bookstore>
```

SAMPLE DTD WITH POINTERS

```
<!DOCTYPE Bookstore [
   <!ELEMENT Bookstore (Book*, Author*)>
   <!ELEMENT Book (Title, Remark?)>
   <!ATTLIST Book ISBN ID #REQUIRED
                  Price CDATA #REQUIRED
                  Authors IDREFS #REQUIRED>
   <!ELEMENT Title (#PCDATA)>
   <!ELEMENT Remark (#PCDATA | BookRef)*>
   <!ELEMENT BookRef EMPTY>
   <!ATTLIST BookRef book IDREF #REQUIRED>
   <!ELEMENT Author (First_Name, Last_Name)>
   <!ATTLIST Author Ident ID #REQUIRED>
   <!ELEMENT First_Name (#PCDATA)>
   <!ELEMENT Last_Name (#PCDATA)>
]>
```

XML FOR SAMPLE DTD WITH POINTERS

```
<Bookstore>
   <Book ISBN="ISBN-0-13-713526-2" Price="100" Authors="JU JW">
      <Title>A First Course in Database Systems</Title>
   </Book>
   <Book ISBN="ISBN-0-13-815504-6" Price="85" Authors="HG JU JW">
      <Title>Database Systems: The Complete Book</Title>
      <Remark>
         Amazon.com says: Buy this book bundled with
         <BookRef book="ISBN-0-13-713526-2" /> - a great deal!
      </Remark>
    </Book>
    <Author Ident="HG">
      <First_Name>Hector</First_Name>
      <Last_Name>Garcia-Molina</Last_Name>
   </Author>
   <Author Ident="JU">
      <First_Name>Jeffrey</First_Name>
      <Last_Name>Ullman</Last_Name>
   </Author>
   <Author Ident="JW">
      <First_Name>Jennifer</First_Name>
      <Last_Name>Widom</Last_Name>
   </Author>
</Bookstore>
```

XSD

- XML Schema.
- Very broad standard to validate XML.
- **Provides a grammar that can be used to define:**
  - Elements
  - Attributes
  - Nesting
  - Ordering
  - Number of occurrences
  - Data types
  - Keys
  - Pointers (**typed**)
  - …
- XSD is out of the scope of this unit. We just provide an example so you can recognize it when you see it.

SAMPLE XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Bookstore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" type="BookType“ minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="Author" type="AuthorType" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:key name="BookKey">
    <xsd:selector xpath="Book" /><xsd:field xpath="@ISBN" />
  </xsd:key>
  <xsd:key name="AuthorKey">
    <xsd:selector xpath="Author" /><xsd:field xpath="@Ident" />
  </xsd:key>
  <xsd:keyref name="AuthorKeyRef" refer="AuthorKey">
    <xsd:selector xpath="Book/Authors/Auth" /><xsd:field xpath="@authIdent" />
  </xsd:keyref>
  <xsd:keyref name="BookKeyRef“ refer="BookKey">
    <xsd:selector xpath="Book/Remark/BookRef" /><xsd:field xpath="@book" />
  </xsd:keyref>
</xsd:element>
```

ADVANTAGES AND DISADVANTAGES OF USING DTD/XSD

Advantages

- Applications can assume that there's an underlying structure to the document.
- CSS/XML can be used safely.
- It's easier to write documentation when there's a fixed structure.
- All other advantages of strong typing.

Disadvantages

- A well-formed XML is easier to modify and more flexible.
- DTD/XSD files can end becoming too big and cumbersome.
- All other disadvantages of weak typing.

XPATH AND XQUERY

NAVIGATING AN XML

- We must think of an XML file as a tree. XPath is an standard that allows us to navigate such tree. XPath takes an XML document or stream as input.
- This section uses the supplementary material marked as “02-XPath”, which contains a sample XML and several examples of how to use XPath. Running these examples in a tool like **Kernow** is the best way to learn.
- **Basic XPath building blocks:**
  - `/`: Separator
  - `//`: Me and all my children
  - `/TagName`: Tags
  - `/@AttributeName`: Atributtes
    - To get its value we write: `/data(@AttributeName)`
  - `|`: OR, used with parenthesis
  - `*`: Wildcard

XPATH CONDITIONS

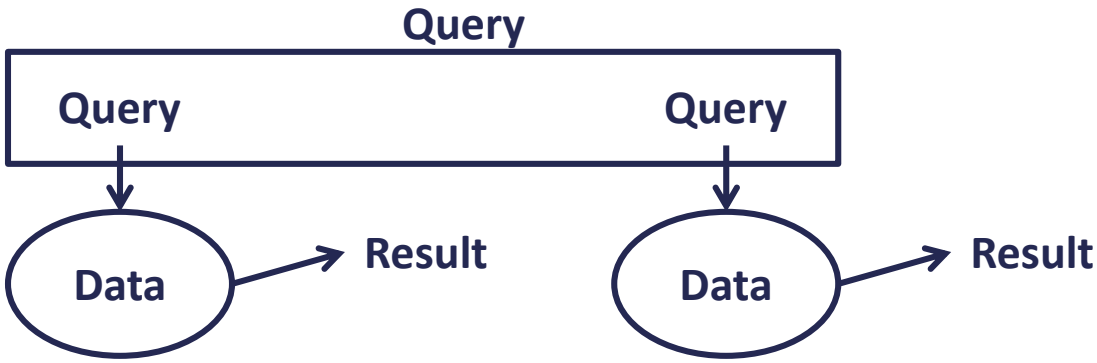
- **Conditions are used to filter tags:**
  - `[]`: Used to separate conditions
    - Could be nested to group conditions
    - They include an implicit “/”
  - `[TagName]`: Existence
  - `<, >, =, !=`: Comparators
  - `and`: logical AND, to link conditions
  - `or`: logical OR, to link conditions
  - `[number]`: Counter

XPATH NAVIGATION AXES AND FUNCTIONS

- XPath contains 13 **navigation axes**. These are some we'll use:
  - `parent::`
  - `preceding-sibling::`
  - `following-sibling::`
  - `descendents:`
  - `self:`
- XPath has several **functions**. These are some we'll use:
  - `contains(element, "text")`
  - `name()`
  - `count()`

XQUERY

- **XQuery** is a language used to make queries over an XML.
- XPath is actually a subset of XQuery.
- It's similar in concept to SQL.
- Queries can be nested.
- XQuery is out of the scope of this course.



XSLT

XSLT

- **XSL:** *Extensible Stylesheet Language*: Initial version.
- **XSLT:** *XSL (with) Transformations*: With some improvements.
- XSLT, unlike XPath, is written using XML. It's used to build templates.
- XSLT sees the document as a collection of nodes:
  elements, attributes, text, comments…
- It's useful to find and replace parts of a XML document (using XPath).
- It can be used recursively.
- Uses structures typical of programming languages:
  - Conditionals: (if-else)
  - Iterators: (for-each)
- While using it, we should put special care with:
  - Strange behaviors with white spaces.
  - Implicit priorities in templates.
- **In this section we'll learn how to use XSLT to turn a XML into a HTML.** It uses the supplementary material marked as "03-XSLT", which contains a XML and several examples of how to use XSLT to turn a XML into a HTML. Running these examples in a tool like **Kernow** is the best way to learn.

XSLT ELEMENTS (I)

- **<xsl:template>**
  - To build templates.
  - What's inside is what's written as output.
  - Can also be used to discard data.
  - The **match** attribute targets a node or set of nodes of the XML.
  - The value of match is an XPath expression.
  - match="/" covers all the document.
  - match="text()" covers all text, but no tags or attributes.
  - match="*|@*|text()" covers all text, but processes each entity independently.
  - Be careful with template ordering.
- **<xsl:value-of>**
  - Gets the value of a node.
  - The **select** attribute specifies what's extracted.
  - The value of select is an XPath expression.

XSLT ELEMENTS (II)

- **<xsl:for-each>**
  - Iterates over a set of elements.
  - The **select** attribute specifies what's iterated.
  - The value of select is an XPath expression.
- **<xsl:sort>**
  - Used inside a for-each.
  - Sorts the elements.
  - The **select** attribute specifies the ordering criterium.
  - The value of select is an XPath expression.
- **<xsl:if>**
  - Imposes a condition to select an element or not.
  - It's used inside a for-each.
  - The **test** attribute specifies the condition.

XML AND JAVA

XML & UML

- **XML is useful for:**
  - Interoperability between different platforms.
  - Transmitting information *(streaming).*
  - Representation of tree-structured data.
- **UML is useful for:**
  - High-level design.
  - Data management in object-oriented languages.
  - Visual representation of data.
- We want to be able to **transform data from XML to UML and vice versa**.
  - So we can use the solution best-tailored to each situation.
  - We'll learn how to make this transformation using Java since:
    - Getting from UML to Java is trivial.
    - Can be done in a similar way in all other object-oriented languages.

XML & JAVA

- There’re several solutions that allow for the management of XML data in Java.
- There’re two approaches:
  - **Process the XML directly:**
    - Similar to JDBC.
    - **JAXP**: Java API for XML Processing.
  - **Translate between XML documents and Java objects:**
    - Similar to JPA.
    - **JAXB**: Java Architecture for XML Binding.
    - We’ll learn how to use this alternative.
- JAXP and JAXB are the most popular XML Java libraries, but there are many others.
- We can also use Java to invoke the already studied XML standards, like XSLT.

JAXB

- **JAXB:** Java Architecture for XML Binding
- Included in the standard JDK since version 6.
- Allows us to perform two operations:
  - **Marshalling:** Turn Java objects into XML documents.
  - **Unmarshalling:** Turn XML documents into Java objects.
- We have to annotate the Java classes that will represent the data contained in the XML documents.
- **JAXB uses the following annotations:**
  - *@XmlRootElement:* java.xml.bind.annotation.XmlRootElement
  - *@XmlElement:* java.xml.bind.annotation.XmlElement
  - *@XmlElementWrapper:* java.xml.bind.annotation.XmlElementWrapper
  - *@XmlAttribute:* import java.xml.bind.annotation.XmlAttribute
  - *@XmlType:* java.xml.bind.annotation.XmlType
  - *@XmlTransient:* java.xml.bind.annotation.XmlTransient