XML

A character-oriented standard for data exchange between heterogeneous systems

Standards for data exchange in distributed environments

- They typically define:
  1. A language to define ***abstract* data types**
  2. "Neutral" (system independent) data representations for any abstract data type that can be defined by the language

     A data representation entails a mechanism that data receivers must use in order to correctly decode/separate data
     - **Often, receivers must know the (abstract) type of the data they will receive, in order to be able to decode/separate them**
- Examples: ASN.1, XDR, CORBA CDR, XML, JSON
- XML and JSON are becoming the most widely used standards, not only for distributed environments

The main XML features

- Data representations are **character-oriented**
  - Human readable & machine readable
  - Memory/bandwidth requirements not optimized
- Data take the form of "**documents**", similar to HTML documents
- Data incorporate information about their type
  - receivers don't need to know data types in advance

XML (eXtensible Markup Language)

- A language that enables the formal description of **markup languages** (a *meta-language*)
- Derived from SGML (Standard Generalized Markup Language)
- Developed by W3C (open standard)
  - XML 1.0 (fifth edition) November 2008
  - XML 1.1 (second edition) August 2006

SGML

- **markup language**: language for the description of documents, based on annotations (markups) that may describe *structural*, *presentation*, and *semantic* aspects of a document
- The markup languages that can be described by SGML are **device/system independent**
- SGML enables the description of syntactic/structural aspects of markup languages (not their semantics)
  - **How markups are written and how they are distinguished from text and from each other**
  - **What markups are possible or required in different parts of the document,**
  - …

SGML

- SGML defines
  - The general syntax of all the markup languages that it can describe:
    - **a common syntactic base**
    - **declarative (non procedural) markup style**
    - …
  - A formalism (**DTD - Document Type Definition**) that enables the description of a specific markup language
- In practice:
  - SGML defines a general (super) markup language.
  - The specific markup languages that can be described by SGML are subsets of the general language.

SGML Terminology

- **SGML Document** (or document instance)
  - a data object that can be described by the general markup language
  - takes the form of a text conformant to the general SGML syntactic rules, including markups, textual data and a reference to a DTD
- **SGML Application**
  - a markup language described by SGML.

Example of reference to a DTD (HTML document)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>LOGIN</title>
<meta http-equiv=content-type content="text/html;
charset=iso-8859-1">
<meta content="mshtml 6.00.2800.1170" name=generator>
<link href="/images/_site/swas.css" type=text/css rel=stylesheet >
<link href="/images/_site/Show_TableDef.css" type=text/css
      rel=stylesheet >
<link href="/_library/styles.css" rel="stylesheet" title="styles"
      type="text/css">
</head>
<body marginheight=1 align=top border=0 topmargin=1 >
…
```

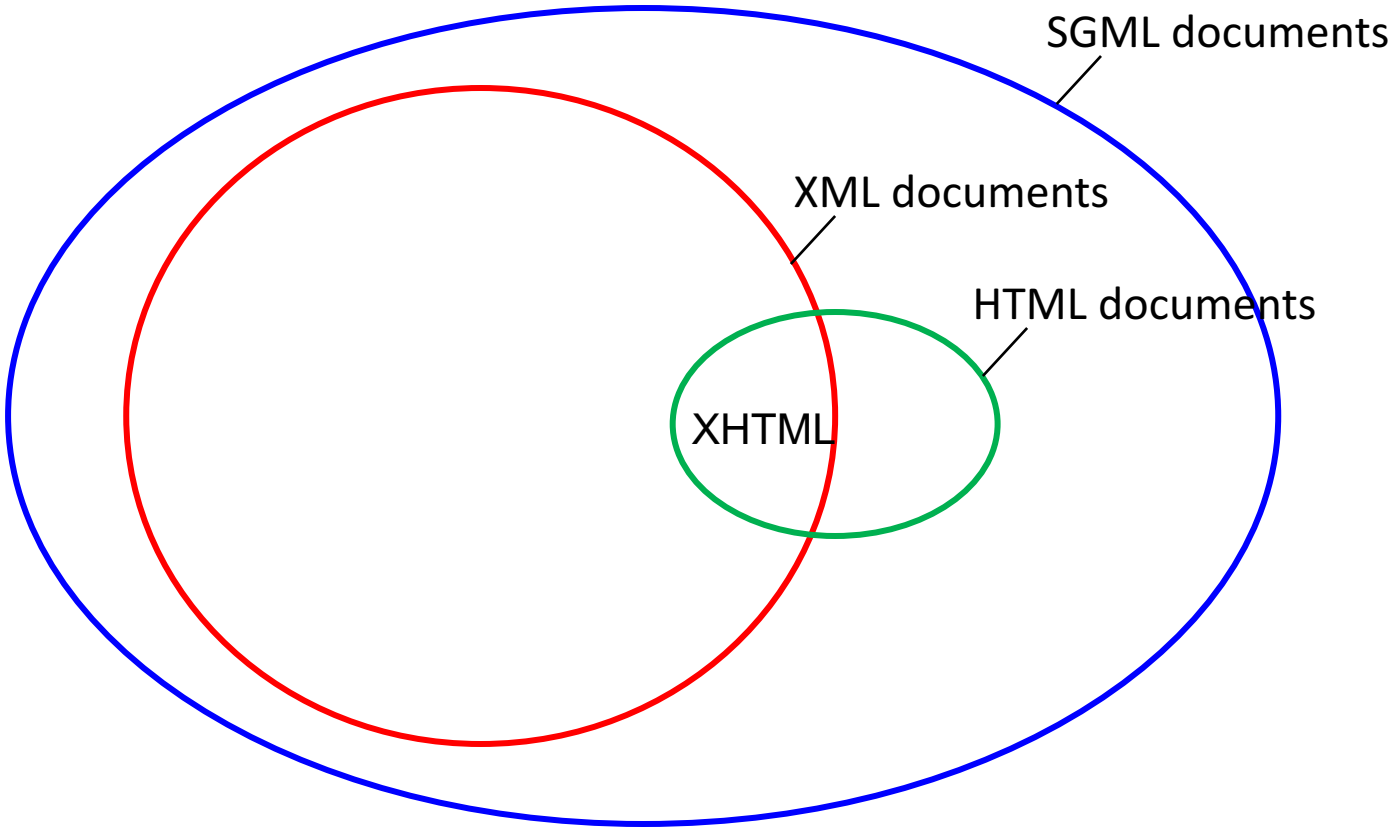
XML

- XML was born with the aim of being an SGML …
  - …directly usable on the internet (via HTTP)
  - …largely open and compatible
  - …directly and simply usable by applications

=> XML incorporates only the simplest aspects of SGML

- Formally, XML is a subset of SGML (an application profile)

SGML, HTML and XML documents



A sample XML Document

```
<?xml version="1.0"?>
<bibliography>
  <article>
    <author> J. W. Cooley </author>
    <author> J. A. Tukey  </author>
    <title> An Algorithm for Machine Computation of Complex FFT
    </title>
    <journal volume="19" number="April 1965"> Math. Computation
    </journal>
  </article>
  <article>
    <author> T. G. Stockham </author>
    <title> High speed convolution and correlation </title>
    <proc year="1966"> Spring Joint Computer Conference </proc>
  </article>
  <book>
    <author> D. A. Chappel </author> <author> T. Jewell </author>
    <title> Java Web Services </title>
    <publisher> Hops Libri </publisher>
  </book>
</bibliography>
```

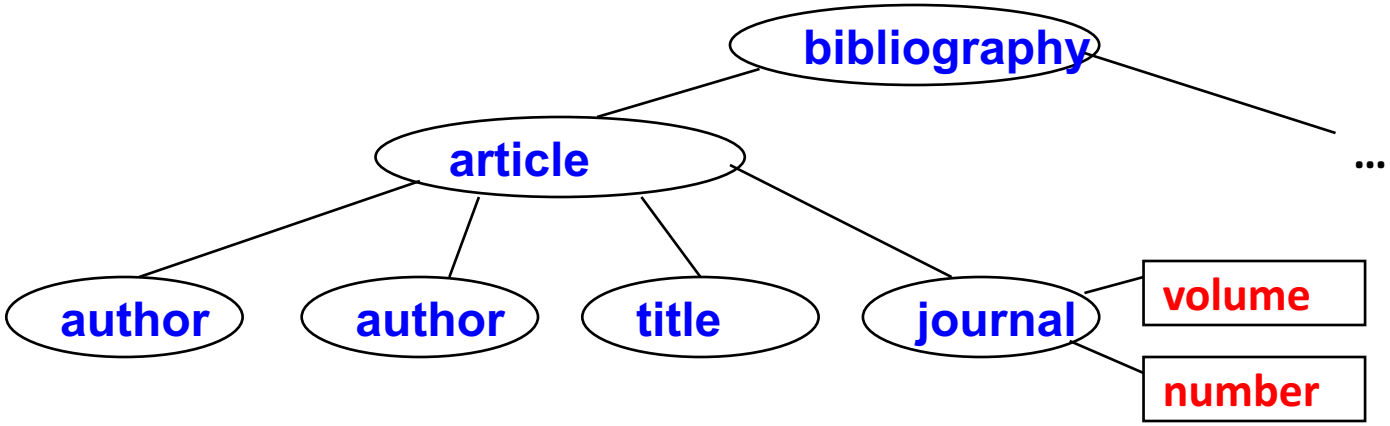
The document concept

- A **document** includes two aspects:
  - The document **contents**
    - contents **structure/syntax** (the organization/format of the contents)
    - contents **semantics** (the meaning of the contents)
  - The document (presentation) **style**
- With CSS (Cascading Style Sheets), style can be separated from contents
- With XML, contents **structure/syntax** can be explicitly described

Logical Organization of an XML Document

- The **contents** of an XML document have *tree structure*
  - Each sub-tree (or node) is an **element**
  - Elements can have **attributes**
  - Elements and attributes carry **data** (text)
- An XML document may include
  - declarations (e.g. to specify the reference DTD)
  - processing instructions
  - comments

```
<?xml version="1.0"?>
<bibliography>
  <article>
    <author> J. W. Cooley </author>
    <author> J. A. Tukey  </author>
    <title> An Algorithm for Machine Computation of Complex FFT
    </title>
    <journal volume="19" number="April 1965"> Math. Computation
    </journal>
  </article>
  …
</bibliography>
```



Physical Organization of an XML Document

- A whole XML document can be distributed on different storage units called **entities**
- An entity can reference other entities
- In any case there is a root entity (**`document entity`**) not referenced by other entities
- For simplicity, we now consider only single entity documents

General Syntax of XML Documents

- XML documents follow the general SGML syntax.
- A text is a **well formed** XML document if it follows the general SGML syntax **and** obeys to some additional rules. Main rules:
  - Each non-empty element is delimited by an initial **and** a final tag
  - There is a single root element (i.e. element that contains all the other elements)
  - Attribute values are always enclosed in quotes
  - Attribute names are unique inside each element (XML languages are case-sensitive)

Syntactic Structures

- A document is made up of **data** (character sequences) and **markups**
- A markup can be:
  - the begin/end tag of a (possibly empty) element
  - a reference to an entity (also used for special characters)
  - a comment
  - a DTD declaration
  - an XML declaration
  - a processing instruction

Specific XML Markups: The XML declaration

- Is placed at the beginning of a text to indicate that the text represents an XML document
- Enables the specification of which XML version and character encoding are used for the document, and whether the document has references to external entities
- Syntax:

```
<?xml version="…" encoding="…" standalone = "…" ?>
```

Examples:

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

The DTD Declaration

- Specifies the reference DTD for the document
- Syntax:

```
<!DOCTYPE name DTD >
```

- May contain a reference to an external definition

Examples:

```
<!DOCTYPE simple SYSTEM "simple.dtd">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

- May contain an internal definition

Example:

```
<!DOCTYPE simple [
        <!ELEMENT item (#PCDATA)>
]>
```

Valid XML Documents

- An XML well-formed document is **valid** if it
  - contains a DTD declaration
  - satisfies the constraints expressed by the DTD
- A DTD defines a language (made up of all the valid documents that refer to that DTD)

$\Rightarrow$ An XML is self-contained:

- It contains both data and structural/syntactic rules with which data are organized

DTD

- Is a sequence of **rules** (element declarations and attribute declarations)
- Rules are written in SGML syntax, with some restrictions and extensions

DTD Example

```
<!-- DTD for telephone users and contracts -->
<!ELEMENT directory ((user|contract)*)>
<!ELEMENT user (#PCDATA)>
<!ATTLIST user
              cf              ID                      #REQUIRED
              type            (private|firm)          #IMPLIED
              advertisements  (admitted|unwanted)     "unwanted"
>
<!ELEMENT contract (address?, capabilities?)>
<!ATTLIST contract
              phone           NMTOKEN         #REQUIRED
              name            IDREF           #REQUIRED
              mobile          NMTOKEN         #FIXED "yes"
>
<!ELEMENT address EMPTY>
<!ATTLIST address
                      street   CDATA          #REQUIRED
                      number   NMTOKEN        #IMPLIED
                      town     CDATA          #REQUIRED
                      zip      NMTOKEN        #REQUIRED
>
```

Element Declaration

- Specifies
  - Element name
  - Content model
- Sintax:

```
<!ELEMENT name model >
```

- Model types:
  - **EMPTY** The element must be empty
  - **ANY** Any contents admitted (no check)
  - ***Element*** The element must contain only elements
  - ***Mixed Content*** The element may contain both elements and data

Element Models

They let us specify **name**, **order, optionality** and **multiplicity** of nested elements by a simple grammar:

- A model is either simply an *element* or *a sequence of models* or *an alternative of models*
  - A comma denotes a sequence
  - A vertical bar denotes an alternative

Multiplicity is specified by the postfix operators:

+	the model must occur 1 or more times
*	the model must occur 0 or more times
?	the model must occur 0 or 1 times (is optional)

Examples of Element Models

```
<!ELEMENT meal (course*)>

<!ELEMENT course (first|second|dessert)>

<!ELEMENT fixedPriceMeal (first,second,dessert)>

<!ELEMENT first EMPTY>

<!ELEMENT second EMPTY>

<!ELEMENT dessert EMPTY>
```

```
<!ELEMENT laboratory ( name, head, secretary?,
   (technician|operator)+)>
```

Mixed Models

- In this case, only the names of admitted nested elements can be specified, not their order and multiplicity
- Syntax is coherent with the one of Element Models, but the only possible form is:

```
(#PCDATA | name1 | name2 | … )*
```

Examples:

```
<!ELEMENT onlyData (#PCDATA)>

<!ELEMENT DataAndFonts (#PCDATA | font)*>

<!ELEMENT DataFontsAndColors (#PCDATA | font | color)*>
```

Attribute Declarations

- Each declaration specifies the features of one or more attributes of an element type
- Syntax:

```
<!ATTLIST ElementName AttributeList >
```

- For every attribute, **name**, **value type** and **default declaration** can be specified

Example:

```
<!ATTLIST onlyData
          id      ID                    #REQUIRED
          type    (vector|matrix)       "vector">
```

Value Type Specification

Type	Value	Syntax	Example
String	A string without the special characters < > & ’ ”	CDATA	CDATA
Token	a token or a sequence of tokens	ID IDREF ENTITY NMTOKEN IDREFS ENTITIES NMTOKENS	ID
Enumeration	One of the specified strings	A list of strings separated by |	(Mr|Mrs|Miss)

Meaning of Various Token Types

Type	Value
ID	A name that *uniquely* identifies the element in the whole XML document (single word, start with letter)
IDREF	The ID of an element in the XML document (a *reference* to an element)
ENTITY	The name of an entity declared in the DTD
NMTOKEN	A generic name (single word, without spaces)
IDREFS	A sequence of IDREF tokens
ENTITIES	A sequence of ENTITY tokens
NMTOKENS	A sequence of NMTOKEN tokens

Default Declaration

- Indicates if the attribute is mandatory (required) or what is its default value. It may take 4 different forms:

#REQUIRED	The attribute is compulsory (no default)
"*default*"	The attribute is optional: if absent, the indicated default value is used
#IMPLIED	The attribute is optional, and the default value is undefined: any value can be used if the attribute is absent
#FIXED *default*	The attribute is optional, but fixed: if present, it must have the indicated default value

Example

```
<!ATTLIST     course
              code          ID          #REQUIRED
              name          CDATA       #IMPLIED
              double        (yes|no)    "no"
>
```

Examples of valid start element tags:

```
<course code="A10" name="Spaghetti">

<course code="A10" double="yes">
```

Examples of invalid start element tags:

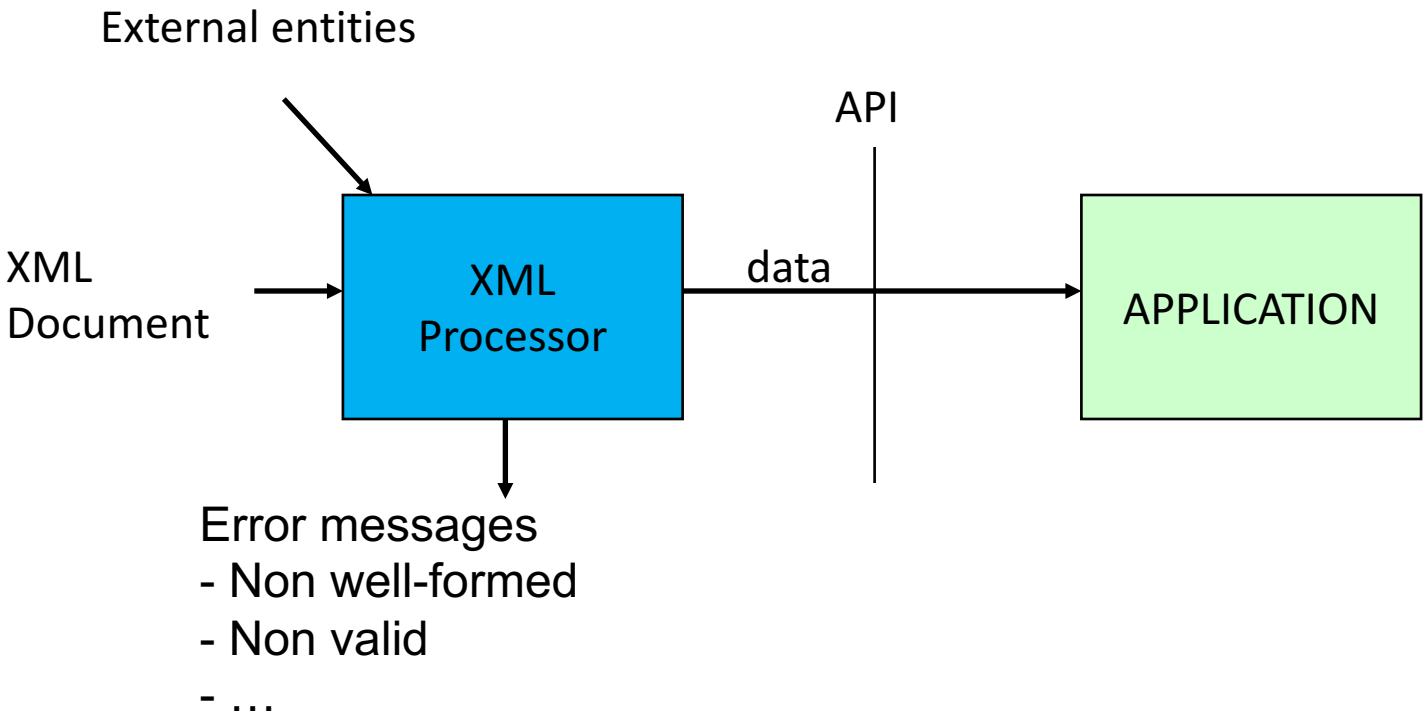
```
<course name="Steak">

<course code="A10" double="nes">

<course>
```

Exercise

- Write a DTD describing the structure of documents that can store the data of a bank account, using the following rules:
  - Each account is characterized by
    - **One and only one account number**
    - **One or more account holders**
    - **A sequence of operations, grouped by year**
  - Each operation is characterized by
    - **date and amount**

    and, optionally, by a **description**
  - Each account holder is characterized by
    - **a name and an address**

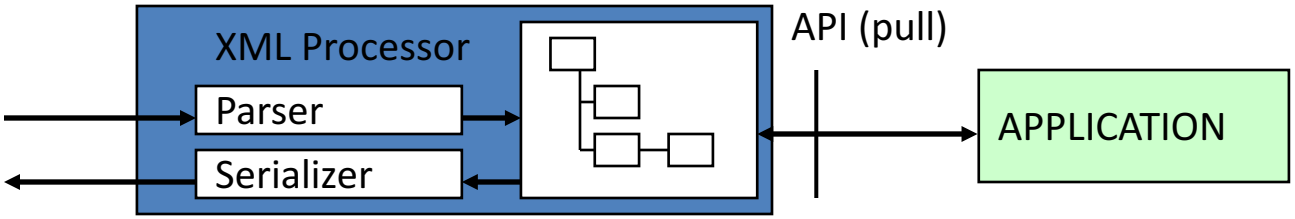
Processing of XML Documents



- **Non Validating Processor** (or Parser): checks only well-formedness (Example: msxml)
- **Validating Processor** : checks both well-formedness and validity (Example: JAXP)

XML Document Processing

DOM:a parser builds in memory the logical tree of the whole XML document. The application accesses the tree by a standard API



SAX: a parser directly passes the recognized tags and data to the application while parsing them

StAX: the application pulls events from or sends events to the XML processor

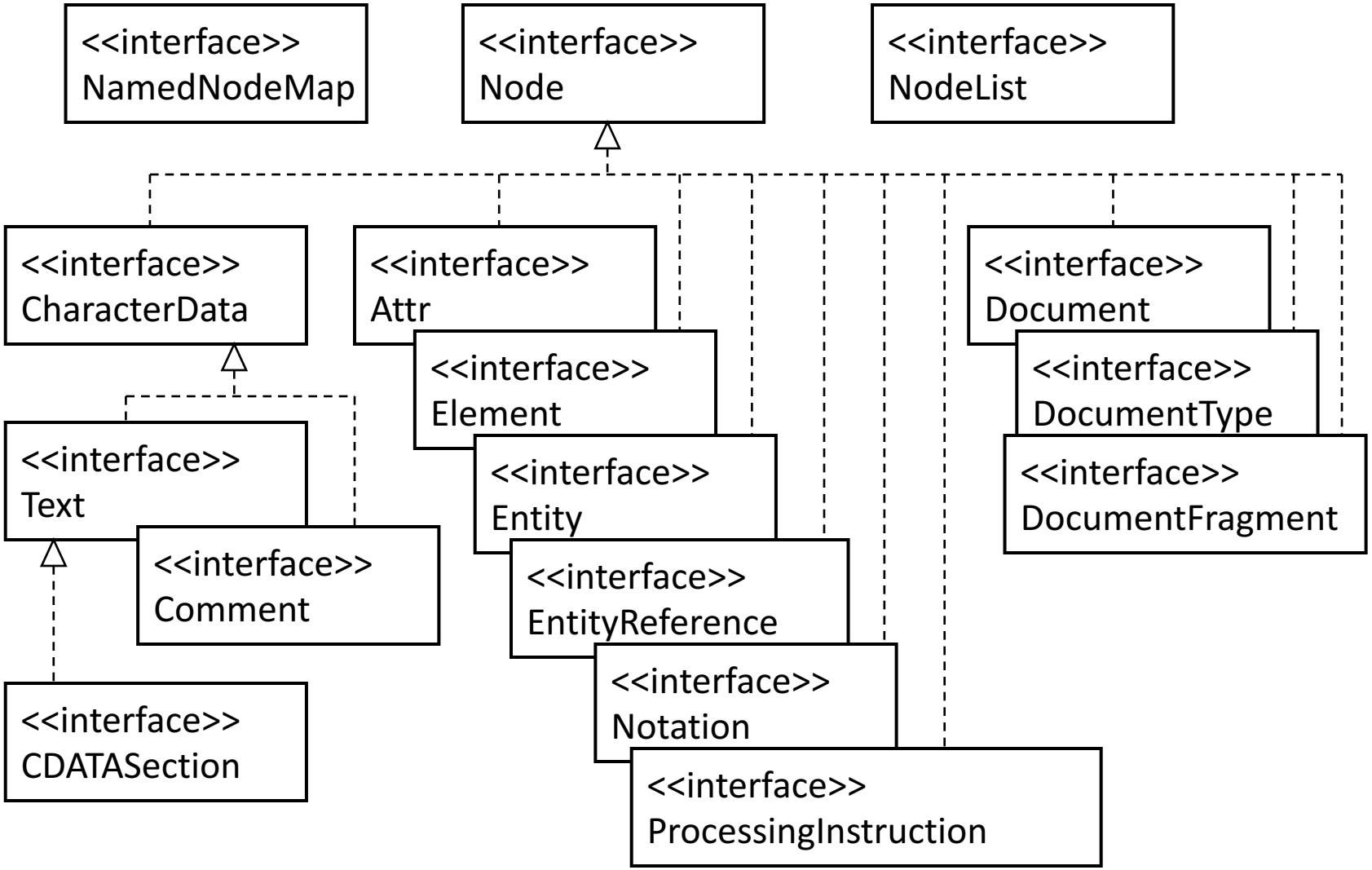
Document Object Model (DOM)

- Developed by W3C, defines a standard way to access the tree-like logical structure of an XML (HTML) document
- Is object-oriented, but abstract enough
  - Not bound to any specific programming language
  - Uses language neutral idiom (IDL CORBA) to define interfaces
  - Implementations are available in C, C++, Java, Perl, …
- Reference documents:
  - W3C Recommendations

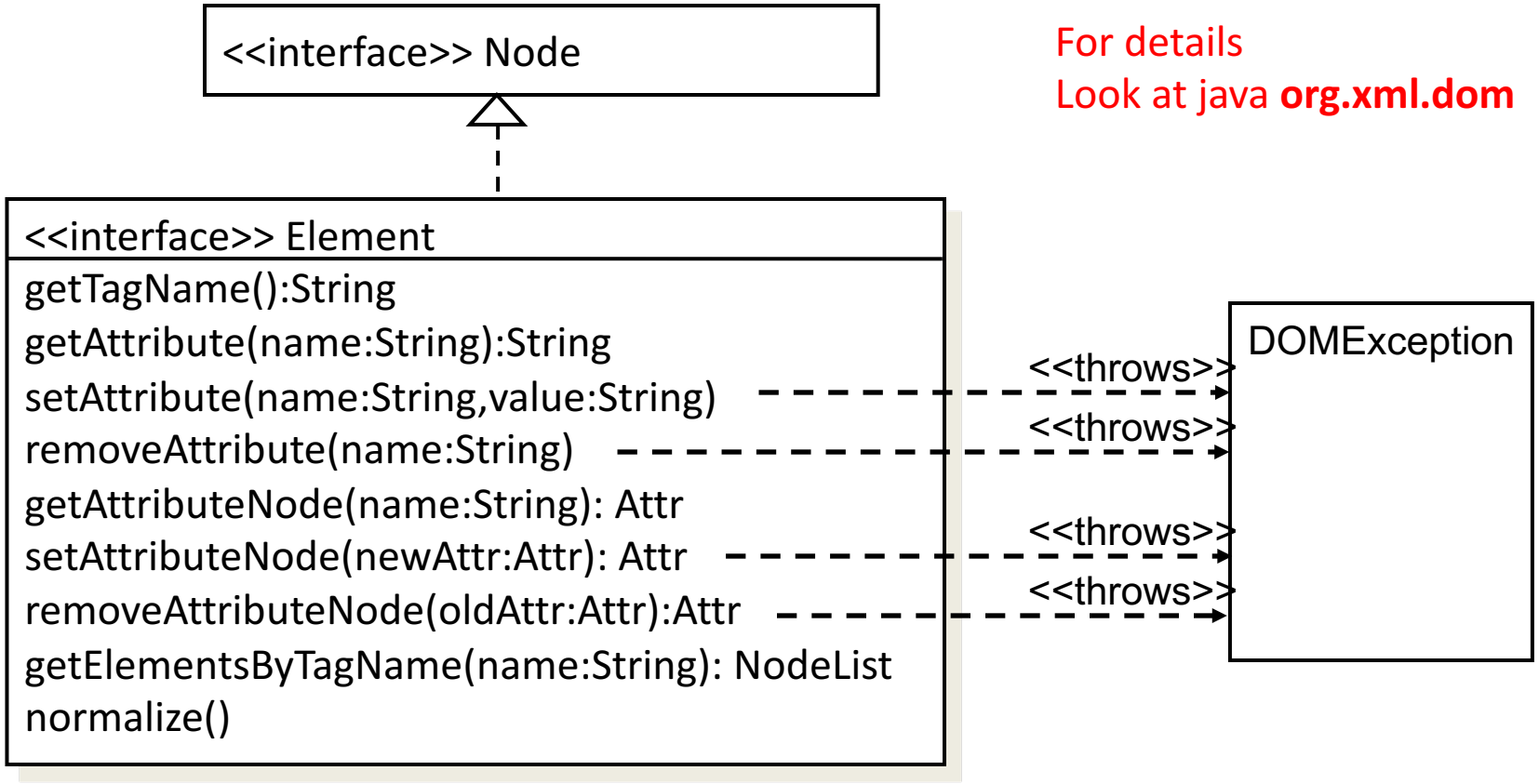
Organization of DOM specifications

- DOM level 1
  - Base functionaliites (simple navigation and document manipulation)
- DOM level 2
  - Further navigation functions
  - Style sheet manipulation
  - Namespace support
  - Event-driven model
- DOM level 3
  - Support for document validation with DTD and schema
  - Enhanced mechanisms for loading and storing XML documents

DOM Level 1 Interfaces



Example: The Element Interface



For details
Look at java **org.xml.dom**

Simple API for XML (SAX)

- Faster than DOM for access to XML contents
- The application must expose some methods which are called by the parser when certain events occur
- Event examples:
  - setDocumentLocator — start of parsing
  - startDocument — start of XML document
  - endDocument
  - startElement — start of an XML element
  - endElement
  - characters — text read

Streaming API for XML (StAX)

- Event-based, pull-parser replacement of SAX
- Can be used for reading and writing XML documents (bidirectional)
- Programming is simpler than with SAX (pull mode)

DOM/SAX/StAX Comparison

	DOM	SAX	StAX
Processing speed	LOW	HIGH	HIGH
Memory occupancy	HIGH	LOW	LOW
Data access ease	HIGH	LOW	MEDIUM
Reading	YES	YES	YES
Writing	YES	NO	YES
Random access	YES	NO	NO

Presentation of XML documents

- Presentation of an XML document is specified separately
- It can be specified using either of:
  - **Cascading Style Sheet (CSS)**
  - **EXtensible Stylesheet Language (XSL)**
- Presentation style is associated with the document by a processing instruction (placed before the root):

```
<?xml:stylesheet type="text/css" href="mystyle.css"?>

<?xml:stylesheet type="text/xsl" href="mystyle.xsl"?>
```

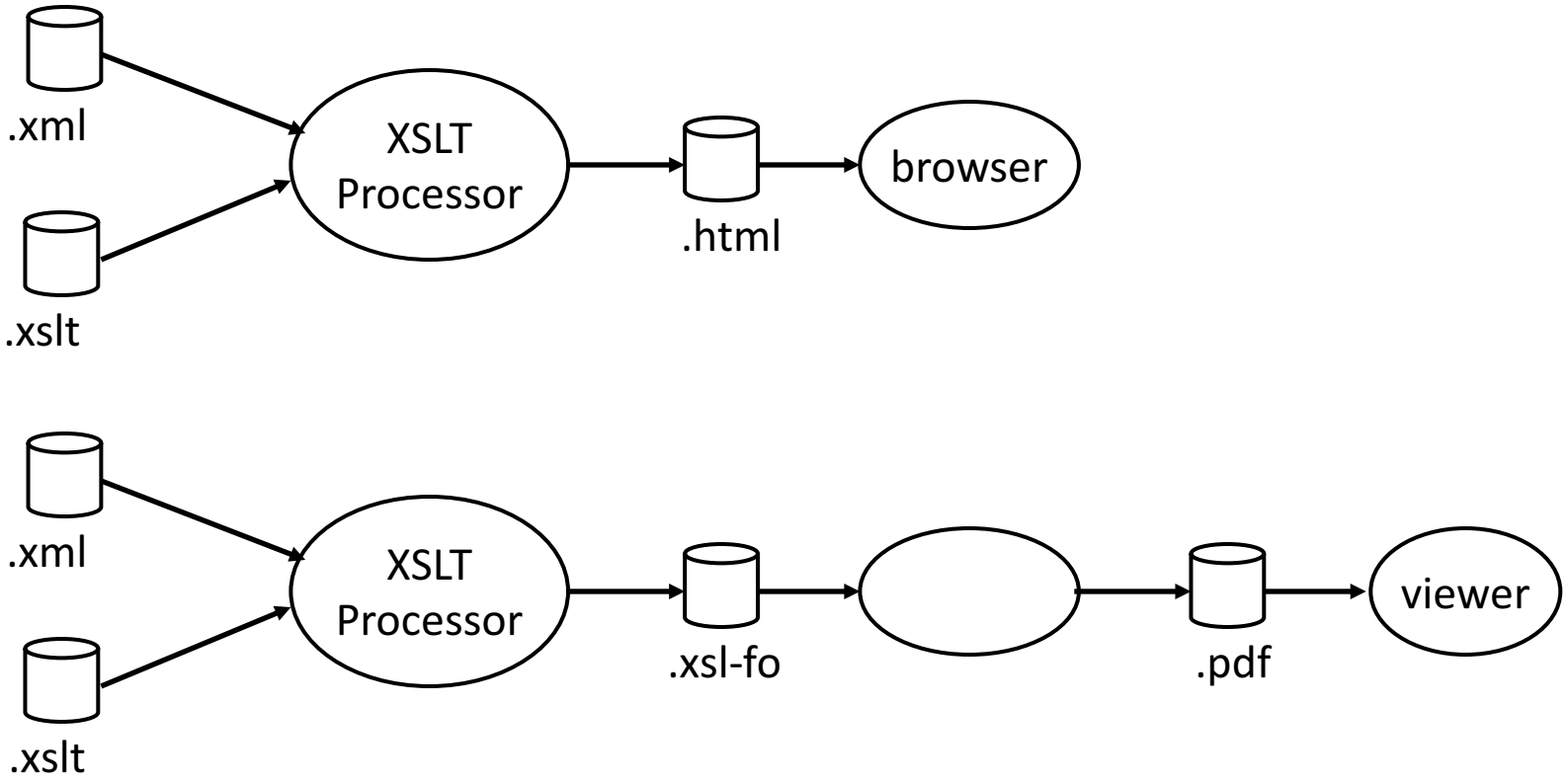
XSL
(eXtensible Stylesheet Language)

- Is a language designed to specify **formatting** (i.e. presentation) of XML documents
  - More advanced than CSS
- Is made up of :
  - **XSLT** (XSL Transformations)

    XML application that enables the specification of transformations that convert XML documents into other types of documents
  - **XSL-FO** (XSL Formatting Objects)

    XML application that enables the specification of a document layout (on pages, text blocks, figures, etc.)

XSL use examples

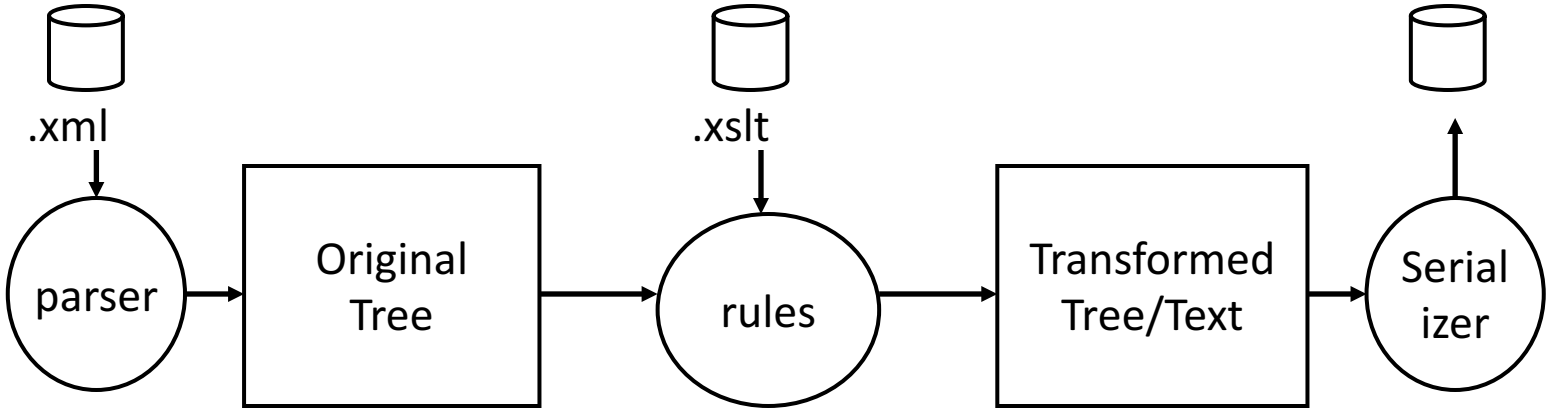


Other Possible Uses of XSLT

- XSLT is indeed a real alternative method for XML document processing
  - XML-XML transformations
  - Report extraction from large documents
  - …

XSLT

- A transformation is described in *declarative* style:
  - Transformation **rules** are specified using “templates”, which are models of parts of the document with associated transformed text
  - Each time a template is matched in the input document, the corresponding transformed text is written to output



XSLT Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <head><link rel="stylesheet" href="style.css"/></head>
  <body>
    <h2>My Articles</h2>
    <table>
    <tr>
      <th>Title</th>
    </tr>
    <xsl:for-each select="bibliography/article">
    <tr>
      <td><xsl:value-of select="title"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body></html>
</xsl:template>
</xsl:stylesheet>
```