

# Enterprise Application Integration

## 1. Basic XML

### 1.1. Introduction to XML

2014/2015



Filipe Araújo  
Informatics Engineering Department  
University of Coimbra  
[filipius@uc.pt](mailto:filipius@uc.pt)  
(slides by Paulo Marques)

## Motivation

- Computers typically need to exchange information between **incompatible systems**
  - **Hardware Incompatibility**  
e.g. PC vs. MAC: different representations for integer and floating point numbers
  - **Data Incompatibility**  
e.g. MS Word vs. Adobe PDF (the information is stored using a proprietary format. This implies that there must exist converters between different formats and that the formats are actually documented.)

2

## Java is not the solution...

- Java and simple agnostic formats are not a solution
  - E.g. Java Serialization Format, Sun XDR, CORBA CDR
- When Java serializes an object to disk, it's stored in a hardware-independent format
  - But, how to use that data in... C++, MS Word, etc?
  - Applications don't understand Java's native formats!
- Sun XDR, CORBA CDR are hardware independent formats but...
  - They were thought for transmitting data through the network, not to store data in disk
  - In 100 years, how can I be sure to be able to access the data?
  - Dichotomy between **storing data** and **transmitting data**
- Databases have the same problem
  - Data is stored in proprietary formats

3

## Business Perspective

- Business Systems need documents that can be read and understood by **people and machines**
  - Well... it's possible to read ASCII from any application. That's one of the reasons it has been chosen for HTML and web pages.
- Documents need to be sufficiently structured such that their information and data can be **automatically processed**
  - Well... HTML can be seen and edited using many different applications. Tags guarantee some structure.
- **So: Why isn't HTML the solution?**

4

## Why going beyond HTML

- HTML is only a standard for how to render **textual information** that must be shown **visually**
  - Tags only define visual appearance. E.g: bold (<b>), italic (<i>)
  - We need something for the data, not how the data is shown.
  - Some data does not have an immediate visual representation (e.g. sound)
- HTML lacks the capability of saying what the data means (lack of meta-information)
  - In databases that's achieved by using a DB *schema*
  - It's necessary to know what each data item represents
  - It's necessary to know the relationship between data items

5

## EDI

- **EDI = Electronic Data Interchange**
- Used for many years and standardized as ANSI X12. (Still quite used!)
- Explicitly thought for commercial data interchange between different business partners
- Complex and Specific
  - "For example an EDI 940 ship-from-warehouse order is used by a manufacturer to tell a warehouse to ship product to a retailer. It typically has a ship to address, bill to address, a list of product numbers (usually a UPC code) and quantities. It may have other information if the parties agree to include it."

6

## An Example of an EDI document

```
ISA*00*      *00*      *01*123454321  *01(Continued)
*012341234   *031016*2359*U*00401*987600111*0*P*:
\GS*RA*123454321*012341234*031016*2359*987600111*X*004010
\ST*820*987600111
\BPR*C*77.77*C*ACH*CTX*01*234056789*DA*0099109999*(Continued)
*123454321*01*045678099*DA*1008973899*031016
\TRN*1*0310162359
\REF*AA*EDI6
\N1*PR*WHIZCO OF AMERICA INC
\N3*55 MEGAPLEASANT ROAD*SUI TE 999
\N4*SUPERVILLE*NY*10954
\N1*PE*YOWZACO
\ENT*1
\RM R*AP*1111111111111111*PO*11.11
\RM R*AP*2222222222222222*PO*22.22
\RM R*AP*4444444444444444*PO*44.44
\DTM*055*031016
\SE*000000014*987600111
\GE*1*987600111
\IEA*1*987600111\
```

7

## XML = eXensible Markup Language

- Substitutes EDI solving many of its problems
- Subset of SGML (*Standard Generalized Markup Language*)
  - Tag-based
  - Structured (documents are seen as a tree)
  - Extensible (the tags are not pre-defined and fixed)
  - Covers both **Data** and **Meta-information**
- Independent of storage and transmission mechanisms
  - e.g. you can send it by email, ftp, you can archive it on a database or a text-file.
  - Data is encoded in ASCII/UNICODE (e.g. UTF-8)
- Readable by humans and editable in any tool
- Can be automatically validated
  - Note: only syntax and structure

8

## Small example...

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```

9

## Small example...

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```

Prolog

Element

(root element)

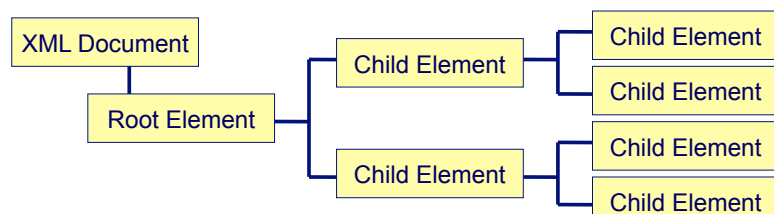
Attribute

Data (Information)

10

## Structure of an XML file

- **Prolog:** First line of the XML file. Specify that's XML!
  - `<?xml version="1.0" encoding="UTF-8"?>`
  - Contains definitions that apply to all the document (e.g. version, encoding, DTD)
- There's a single **root element** that encapsulates all others. The elements below are called *nodes* or *child nodes*.
- It's **necessarily hierarchical**.



11

## Well-formed documents

- The prolog is compulsory
- All elements must have an opening tag and closing tag
  - `<title>Screaming Fields of Sonic Love</title>`
  - Elements without data (e.g. `<OutOfStock></OutOfStock>`) can be represented by a single tag (e.g. `<OutOfStock/>`)
  - XML is case-sensitive. E.g. `<title>` is different from `<TITLE>`
  - Attributes must be between quotes. E.g. `<cd id="0002">`
  - Elements which name starts with "?" represent special processing instructions which are application specific.
- Elements must have a correct sequencing following a tree structure
  - `<cd><title></cd></title>` is incorrect!
  - There must be a root element
- **XML Identifiers**
  - Cannot start by numbers or punctuation signs
  - Can contain letters and numbers but not spaces
  - Cannot start by "XML", "xml", etc.
  - ":" is reserved for namespaces

12

## Elements vs. Attributes

- In many cases information can be represented either as elements or attributes
  - There isn't a clear rule on when to use each
- Rules of thumb
  - Elements can have hierarchy, attributes cannot
  - Elements can store multiple values, attributes cannot
  - Identifiers are normally attributes

```
<cd id="0002">
  <title>Uh Huh Her</title>
  <artist>PJ Harvey</artist>
  <year>2004</year>
</cd>
```

vs.

```
<cd>
  <id>0002</id>
  <title>Uh Huh Her</title>
  <artist>PJ Harvey</artist>
  <year>2004</year>
</cd>
```

13

## XML Namespaces

- Imagine that you have to create a single XML file from two different sources. E.g. create a single unified catalog of products

cd\_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
</catalog>
```

book\_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <book id="0001">
    <title>1984</title>
    <author>George Orwell</author>
    <year>1949</year>
  </book>
</catalog>
```

Problem: The same elements are used to identify data with different semantics.

14

## XML Namespaces (2)

- Imagine that you have to create a single XML file from two different sources. E.g. create a single unified catalog of products

cd\_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cdns:catalog xmlns:cdns="http://cdstore.com/cd_catalog">
  <cdns:cd id="0001">
    <cdns:title>Screaming Fields of Sonic Love</cdns:title>
    <cdns:artist>Sonic Youth</cdns:artist>
    <cdns:year>1995</cdns:year>
  </cdns:cd>
</cdns:catalog>
```

book\_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bns:catalog xmlns:bns="http://bookstore.com/book_catalog">
  <bns:book id="0001">
    <bns:title>1984</bns:title>
    <bns:author>George Orwell</bns:author>
    <bns:year>1949</bns:year>
  </bns:book>
</bns:catalog>
```

15

## XML Namespaces (3)

- It's not necessary to prefix all elements
  - Child elements inherit from their parents if a default namespace exists:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="http://cdstore.com/cd_catalog">
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
</catalog>
```

- Otherwise, the namespace must be set explicitly for all children
- Check this:  
<http://geekswithblogs.net/dmillard/archive/2004/10/20/12935.aspx>
- The URI specified using `xmlns` does not have to contain anything valid. It's only used as identifier (key).
  - Even so, it's common that it points to a real web page.
- **One important aspect** of all this is that different organizations standardize different XML formats for different domain applications (XML Schemas).
  - XML Schemas ↔ XML Namespaces
  - E.g. XSD, XSL, MathML, BPML4WS, etc.

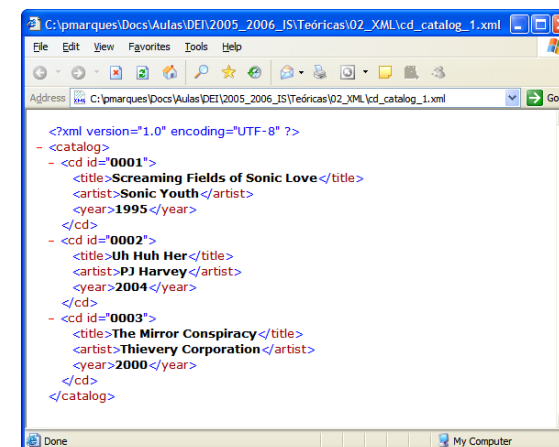
16

## Example of XML and Namespaces

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```

17

## Example of XML and Namespaces (2)



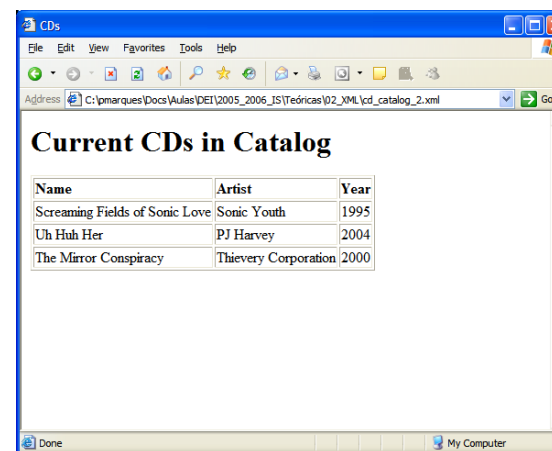
18

## Example of XML and Namespaces (3)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="catalog_to_html.xsl"?>
<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```

19

## Example of XML and Namespaces (4)



20

## catalog\_to\_html.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
  <head><title> CDs </title></head> <body>
    <h1> Current CDs in Catalog</h1>
    <table border="1">
      <tr>
        <td><b> Name </b></td>
        <td><b> Artist </b></td>
        <td><b> Year </b></td>
      </tr>

      <xsl:for-each select="//cd">
        <tr>
          <td><xsl:value-of select="title"/> </td>
          <td><xsl:value-of select="artist"/> </td>
          <td><xsl:value-of select="year"/> </td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
```

## Other relevant issues

- White space is preserved
  - Even so, when processing XML there are several ways of treating white space (normalization, canonical forms)
- CR/LF is converted into LF
- Comments:
  - <!-- This is comment -->
- Entities
  - Certain characters are reserved (e.g. "<")
  - To include them use "&name;". This type of construction is called an entity and it has to be previously defined. E.g:
    - &lt; = "<"
    - &gt; = ">"
    - &copy; = "©"
  - In reality, you can pre-define any string

22

# Enterprise Application Integration

## 1. Basic XML 1.2. Validation – DTD and XSD

2014/2015



Filipe Araújo  
Informatics Engineering Department  
University of Coimbra  
filipius@uc.pt  
(slides by Paulo Marques)

## Validation

- Having a well-formed document does not mean it's valid.
  - How can you tell if a certain element (tag) can be present?
  - How can you tell if a certain element can have a certain attribute?
  - How can you tell if a certain element cannot occur more than once?
- **DTD = Document Type Definition**
  - Original specification which states which elements and attributes a certain XML file can have, their order and number of times they can appear
  - DTD: Specifies if documents are structurally valid
  - The DTD specification is not XML!
  - Does not support datatypes!
- **XML Schema (XSD)**
  - Similar objective to DTDs, but using XML
  - Supports datatypes and advanced validation
  - Currently, the most widely used approach

24

## A simple DTD specification

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog [
  <ELEMENT catalog      (cd*)>
  <ELEMENT cd            (title, artist, year?)>
  <!ATTLIST cd           id CDATA #REQUIRED>
  <ELEMENT title         (#PCDATA)>
  <ELEMENT artist        (#PCDATA)>
  <ELEMENT year          (#PCDATA)>
]>

<catalog>
  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```

25

## Separation between information and meta-information

- Although DTDs can be directly embedded in XML doing so is not a good idea
  - It's important to have a clear separation between information (XML) and meta-information (DTD, XSD)

cd\_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "book_catalog.dtd">

<catalog>
  ...
</catalog>
```

cd\_catalog.dtd

```
<!DOCTYPE catalog [
  <ELEMENT catalog      (cd*)>
  <ELEMENT cd            (title, artist, year?)>
  <!ATTLIST cd           id CDATA #REQUIRED>
  <ELEMENT title         (#PCDATA)>
  <ELEMENT artist        (#PCDATA)>
  <ELEMENT year          (#PCDATA)>
]>
```

26

## DTD Definition – Elements

- A DTD defines a grammar for what definitions are valid in an XML file
- Definitions are recursive.
- Each element is specified using the notation **!ELEMENT**
  - Each element can be simple text (#PCDATA – *Parsed Character Data*) or other elements
  - If an element is composed of other elements, you may specify how many times each can appear:
    - ? = 0 or 1 time
    - + = 1 or more times
    - \* = 0 or more times
    - If you want a specific number of times, you have to do it manually, as a sequence
  - Sequences of elements are defined by name being separated by commas “,”

27

## Going back to the example...

```
<!DOCTYPE catalog [
  <ELEMENT catalog      (cd*)>
  <ELEMENT cd            (title, artist, year?)>
  <!ATTLIST cd           id CDATA #REQUIRED>
  <ELEMENT title         (#PCDATA)>
  <ELEMENT artist        (#PCDATA)>
  <ELEMENT year          (#PCDATA)>
]>
```

- “**catalog**” is the **root element**
- “**catalog**” has **0 or more “cd”**
- Each “**cd**” has, in sequence:
  - ONE** entity “**title**”
  - ONE** entity “**artist**”
  - ZERO OR ONE** entities “**year**”
- A “**title**” is simple **text** (#PCDATA)
- An “**artist**” is simple **text** (#PCDATA)
- A “**year**” is simple **text** (#PCDATA)

28

## Elements (2)

### ■ Defining elements

- Empty elements (e.g. <in\_stock/>) are specified by using EMPTY (e.g. <!ELEMENT in\_stock EMPTY>)
- Elements containing any type of information are specified by using ANY (in general, you should not use them...)
- Alternatives while defining content are specified using pipes. For instance:

```
<!ELEMENT contact (email|phone|fax)+ >
```

«Each element “contact” has one or more  
“email”, “phone” or “fax”»

- It's also possible to specify sequences with alternatives:

```
<!ELEMENT contact (email, (phone|fax)*) >
```

«Each element “contact” has to have an “email” where this  
email can be followed, optionally, by one or more occurrences  
of “phone” and “fax”»

29

## Defining Attributes

- Each attribute of an element is specified using !ATTLIST

```
<!ATTLIST cd id CDATA #REQUIRED>
```

1      2      3      4

1. Name of the entity
2. Name of the attribute
3. Data type
4. Parameterization

- «Each entity “cd” must necessarily have (#REQUIRED) an attribute called “id”, which value is text (#CDATA)»

30

## Defining Attributes (2)

- General format while defining an attribute:

```
<!ATTLIST element_name attribute_name attribute_type parameterization>
```

- E.g. DTD: <!ATTLIST cd is\_available CDATA “yes”>  
XML: <cd id=“0001” is\_available=“no”>
- Typically, the parameterization of an attribute represents a default value (in this case, “yes”). Nevertheless, other keywords can be used (e.g. #REQUIRED)

31

## Attribute Parameterization

| Value        | Explanation   |
|--------------|---|
| “value”      | The default value of the attribute                  |
| #REQUIRED    | The attribute value must be included in the element |
| #IMPLIED     | The attribute does not have to be included          |
| #FIXED value | The attribute value is fixed                        |

Examples at:  
<http://www.w3schools.com/dtd>

32



## Some examples...

- **Default attribute value**
  - DTD: `<!ELEMENT square EMPTY>`  
`<!ATTLIST square width CDATA "0">`
  - Valid XML: `<square width="100" />`
- **#IMPLIED**
  - DTD: `<!ATTLIST contact fax CDATA #IMPLIED>`
  - Valid XML: `<contact fax="555-667788" />` or `<contact/>`
- **#REQUIRED**
  - DTD: `<!ATTLIST person number CDATA #REQUIRED>`
  - Valid XML: `<person number="5677" />`
- **#FIXED**
  - DTD: `<!ATTLIST sender company CDATA #FIXED "Microsoft">`
  - Valid XML: `<sender company="Microsoft" />`
  - Invalid XML: `<sender company="W3Schools" />`
- **Enumerated attribute values**
  - DTD: `<!ATTLIST payment type (check|cash) "cash">`
  - Valid XML: `<payment type="check" />`

33

## Attribute Types (attribute\_type)

| Value        | Explanation                                   |
|--------------|---|
| CDATA        | The value is character data                   |
| (en1 en2 ..) | The value must be one from an enumerated list |
| ID           | The value is a unique id                      |
| IDREF        | The value is the id of another element        |
| IDREFS       | The value is a list of other ids              |
| NMTOKEN      | The value is a valid name token               |
| NMTOKENS     | The value is a list of valid name tokens      |
| ENTITY       | The value is an entity                        |
| ENTITIES     | The value is a list of entities               |
| NOTATION     | The value is a name of a notation             |
| xml:         | The value is a predefined xml value           |

Complete definition and examples at:  
<http://infohost.nmt.edu/tcc/help/pubs/dtd/attr-decl.html>

34

## Entity Definition

- Entities correspond to text abbreviations
  - DTD: `<!ENTITY copyright "(c) Copyright, 2005 – DEI">`  
XML: `<direitos> &copyright; </direitos>`
  - ... in reality: `"&lt;"`, `"&gt;"` are predefined entities in XML
- Entities can either be internal or external
  - Internal entities are defined directly on the current file  
`<!ENTITY copyright "(c) Copyright, 2005 – DEI">`
  - External entities are defined in some other file or URI/URL  
`<!ENTITY copyright SYSTEM "system.dtd">`

35

## But...

- DTDs are not something specified in XML
  - Being XML a specified on how to store data, one would expect that the format of an explicit XML file (DTD) would be an XML file in itself
- A XML Schema corresponds to a specific set of tags that are used to describe metadata (i.e. DTDs)
- The functionality associated to XML Schema (XSD) is similar to the one of DTDs but... in XML!
  - In terms of added-value, the major improvement is the usage of data-types (pre-defined or user-defined) and allowing for a stricter data validation.
  - Most important problem: COMPLEXITY. People normally use tools to write it.

36

## Example of an XSD file (cd\_catalog.xsd)

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="cd" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="cd">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="title" />
        <xsd:element ref="artist" />
        <xsd:element minOccurs="0" maxOccurs="1" ref="year" />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="title" type="xsd:string" />
  <xsd:element name="artist" type="xsd:string" />
  <xsd:element name="year" type="xsd:string" />
</xsd:schema>
```

37

## Example of an XSD file (cd\_catalog.xsd)

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="cd" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="cd">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="title" />
        <xsd:element ref="artist" />
        <xsd:element minOccurs="0" maxOccurs="1" ref="year" />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="title" type="xsd:string" />
  <xsd:element name="artist" type="xsd:string" />
  <xsd:element name="year" type="xsd:string" />
</xsd:schema>
```

38

## How to specify an XSD file?

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="http://cdstore.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://cdstore.com/cd_catalog.xsd">

  <cd id="0001">
    <title>Screaming Fields of Sonic Love</title>
    <artist>Sonic Youth</artist>
    <year>1995</year>
  </cd>
  <cd id="0002">
    <title>Uh Huh Her</title>
    <artist>PJ Harvey</artist>
    <year>2004</year>
  </cd>
  <cd id="0003">
    <title>The Mirror Conspiracy</title>
    <artist>Thievery Corporation</artist>
    <year>2000</year>
  </cd>
</catalog>
```

39

## Another (simple) example: book.dtd conversion

```
<!DOCTYPE book [
  <!ELEMENT book (title,author)>
  <!ATTLIST book category (Fiction|Non-Fiction) #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
]>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<book category="Non-Fiction">
  <title>C# -- Curso Completo</title>
  <author>Paulo Marques</author>
</book>
```

40

## The corresponding Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="book">
    <xsd:complexType>

      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" type="xsd:string"/>
      </xsd:sequence>

      <xsd:attribute name="category" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Fiction" />
            <xsd:enumeration value="Non-Fiction" />
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>

    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

41

## Some important points...

- Each element can be:
  - Simple: It only has text (No children or attributes)
 

```
<xsd:element name="title" type="xsd:string"/>
```
  - Complex: Has children and/or attributes
 

```
<xsd:element name="book"> <xsd:complexType> (...)
</xsd:complexType> </xsd:element>
```
- Standard data types are defined at:
 

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

  - String, Decimal, Integer, Boolean, Date, Time, ...**
- It fully supports everything you can do on a DTD. E.g:
  - ```
<xsd:element name="color" type="xsd:string" default="red"/>
```
  - ```
<xsd:element name="color" type="xsd:string" fixed="red"/>
```
  - ...
  - The same applies to attributes...

42

## But XSD allow for more powerful validations

```
<xsd:element name="age">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Wrong!

<age>120</age>

```
<xsd:element name="car">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="BMW">
      <xsd:enumeration value="Audi">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Wrong!

<car>Mini</car>

```
<xsd:element name="phone">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:pattern value="[0-9]{9}">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Wrong!

<phone>123</phone>

43

## Possible restrictions

| Constraint     | Description   |
|----------------|---|
| enumeration    | Defines a list of acceptable values   |
| fractionDigits | Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero           |
| length         | Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero   |
| maxExclusive   | Specifies the upper bounds for numeric values (the value must be less than this value)                  |
| maxInclusive   | Specifies the upper bounds for numeric values (the value must be less than or equal to this value)      |
| maxLength      | Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero |
| minExclusive   | Specifies the lower bounds for numeric values (the value must be greater than this value)               |
| minInclusive   | Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)   |
| minLength      | Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero |
| pattern        | Defines the exact sequence of characters that are acceptable  |
| totalDigits    | Specifies the exact number of digits allowed. Must be greater than zero                                 |
| whiteSpace     | Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled                   |

44

## It also allows to do advanced data modeling

```
<xsd:element name="employee" type="fullpersoninfo"/>

<xsd:complexType name="personinfo">
  <xsd:sequence>
    <xsd:element name="firstname" type="xsd:string"/>
    <xsd:element name="lastname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="fullpersoninfo">
  <xsd:complexContent>
    <xsd:extension base="personinfo">
      <xsd:sequence>
        <xsd:element name="address" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="country" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

(In fact, there are even tools to map from DB schemas into XML!)

45

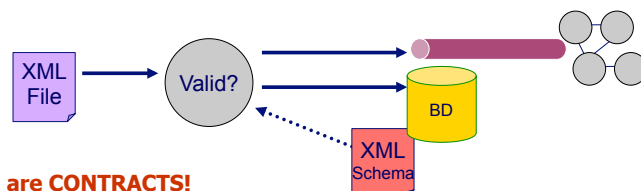
## Key points regarding XSD

- Defined in XML (meta-circularity)
  - Supports general data-types
  - Supports advanced validation
  - It's possible to map relational models using restrictions, references and keys
  - Modular and with support for namespaces
- 
- ... Steep learning curve
  - ... Slow when compared to DTDs

46

## Validation in the “real world”...

- Consider an application which is processing an XML file:
  - Either it is thrown an exception because it finds something that is not expecting (e.g. a certain tag is not present)
  - Or everything processes ok
  - But doing validation with XSD/DTD is SLOW!
- Why use validation with DTD or XSD?
  - In reality, many times applications don't do it.
  - Exceptions and validation are only done at the frontier of systems and at the entrance of databases



- **DTD and XSD are CONTRACTS!**
  - “What is the format of the XML file I need to process?”
  - It's a **formal specification** of the data to process
  - Organizations create standards which specify the schemas used in certain business areas (e.g. OASIS)

47

## OASIS – <http://www.oasis-open.org>



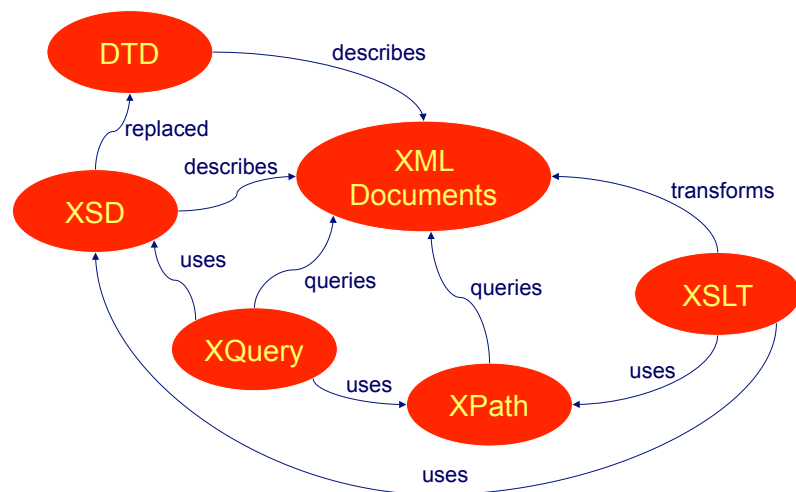
Advancing E-Business Standards Since 1993

- Web Services/SOA
- e-Commerce
- Security
- Law & Government
- Supply Chain
- Computing Mgmt
- Application Focus
- Document-Centric
- XML Processing
- Conformance/Interop
- Industry Domains

- **OASIS e-Government TC**  
OASIS announces our “Second Annual Adoption Forum in Europe” on 17 October 2005 in London, England. More specific details may be found on the event Web site. Join us to hear about the work of the OASIS eGov Technical Committee from a presentation by Harm Jan van Burg. Event link: [http://www.oasis-open.org/events/adoption\\_forum\\_2005/index.php](http://www.oasis-open.org/events/adoption_forum_2005/index.php) Providing a forum for governments to articulate and coordinate requirements for XML- and Web services-based standards
- **OASIS Electronic Procurement Standardization (eProc) TC**  
Researching and developing global e-p
- **OASIS Materials Markup Language (MML) TC**  
Standardizing the exch manufacturing materia
- **OASIS Product Life Cycle (PLC) TC**  
Collaborating on the d (ISO 10303) to support
- **OASIS HumanMarkup TC**  
Using XML to contextually convey cultural, social, kinesics, and psychological intent within communications
- **OASIS International Health Continuum (IHC) TC**  
Providing a forum for the global healthcare community to articulate and coordinate requirements for XML- and Web services-based standards
- **OASIS Open Building Information Exchange (eBIX) TC**  
Enabling mechanical and electrical control systems in buildings to communicate with enterprise applications
- **OASIS Universal Business Language (UBL) TC**  
Defining a com invoices, etc.)
- **OASIS Product Life Cycle Support (PLCS) TC**  
Collaborating on the deployment of an international standard for product data exchange (ISO 10303) to support complex engineered assets from concept to disposal
- **OASIS Translation Web Services TC**  
Automating the translation and localization process as a Web service

48

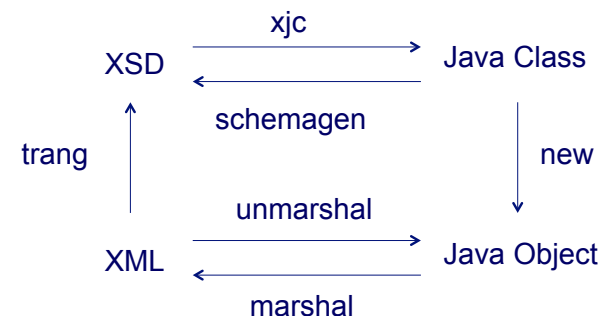
## What have we seen so far?



(c) SOA Systems Inc. -- <http://www.specifications.ws/> (adapted)

49

## JAXB



50

## IMPORTANT NOTICE

YOU ARE FREE TO USE THIS MATERIAL FOR YOUR PERSONAL LERNING OR REFERENCE, DISTRIBUTE IT AMONG COLLEGUES OR EVEN USE IT FOR TEACHING CLASSES. YOU MAY EVEN MODIFY IT, INCLUDING MORE INFORMATION OR CORRECTING STANDING ERRORS.

THIS RIGHT IS GIVEN TO YOU AS LONG AS YOU KEEP THIS NOTICE AND GIVE PROPER CREDIT TO THE AUTHOR. YOU CANNOT REMOVE THE REFERENCES TO THE AUTHOR OR TO THE INFORMATICS ENGINEERING DEPARTMENT OF THE UNIVERSITY OF COIMBRA.

(c) 2009-2014 – Paulo Marques, [pmarques@dei.uc.pt](mailto:pmarques@dei.uc.pt) and Filipe Araújo, [filipius@uc.pt](mailto:filipius@uc.pt)

51