

Curso de Doctorado 2004-05

Arquitecturas de Bases de Datos Web. Ontologías en la Web Semántica

## *Documentos Web Estructurados en XML*

Transparencias basadas en el libro: *A Semantic Web Primer*, G. Antoniou and F. van Harmelen, Springer 2004

Por: C. Costilla

SINBAD-UPM Research Group

<http://sinbad.dit.upm.es>

## Contents of Chapter 2

### *Structured Web Documents in XML*

- 2.1 Introduction
- 2.2 The XML language
- 2.3 Structuring
- 2.4 Namespaces
- 2.5 Addressing and Querying XML documents
- 2.6 Processing
- 2.7 Summary
- Suggested Reading.
- Exercises and Projects

## 2 Structured Web Documents in XML.

### 2.1 Introduction

- 2.1 Introduction
- 2.2 The XML language
- 2.3 Structuring
- 2.4 Namespaces
- 2.5 Addressing and Querying XML documents
- 2.6 Processing
- 2.7 Summary
- Suggested Reading.
- Exercises and Projects

#### ■ Similarities between HTML and XML:

- ✓ HTML is the standard language in which Web pages were initially written. It was derived from SGML (ISO 8879) for the definition of device (and systems) methods of representing information; both human -and machine- readable. HTML is an application of SGML
- ✓ XML is another application of SGML, developed later than HTML.
- ✓ Both languages use tags and they are *markup languages* that permit to write some content and provide information about what role that content plays.
- ✓ In both languages tags may be nested (tags within tags)
- ✓ Both languages were designed to be easily understandable by humans. But, WHAT about machines???

## 2 Structured Web Documents in XML.

### 2.1 Introduction

#### 2.1 Introduction

#### 2.2 The XML language

#### 2.3 Structuring

#### 2.4 Namespaces

#### 2.5 Addressing and Querying XML documents

#### 2.6 Processing

#### 2.7 Summary

#### Suggested Reading.

#### Exercises and Projects

### ■ XML versus HTML:

- ✓ All tags in XML must be closed, whereas in HTML some tags, such as <br>, may be left open.
- ✓ The enclosed content, together with its opening and closing tags, is referred to as an element.
- ✓ The recent XHTML is more in line with XML: any valid XHTML document is also a valid XML document, consequently, tags in XHTML are balanced
- ✓ The problem arise from the fact that the HTML document does not contain structural inf, that is, inf about pieces of the document and their relationships. In contras, the *XML document is far more easily accessible to machines* because every piece of inf is described.
- ✓ Another advantage is that *XML allows the definition of constraints on values*.
- ✓ The HTML representation provides more than the XML representation: the formatting of the document is also described. However, this feature is a weak of HTML (it must specify the formatting, when the main use is to display inf)

## 2 Structured Web Documents in XML.

### 2.1 Introduction

#### ■ XML versus HTML:

✓ XML separates content from formatting. The same inf can be displayed in different ways, without requiring multiples copies of the same content. Moreover, the content may be used for purposes other than display.

✓ Example of HTML text (HTML tags are fixed: list, bold, color, etc):

```
<h2>Relationship force-mass</h2>  
<i>F = M x a</i>
```

✓ Example of XML representation:

```
<equation>  
  <meaning>Relationship force-mass</meaning>  
  <left side>F </left side>  
  <right side>M x a </right side>  
</equation>
```

✓ XML could use different tags from other documents with the same content.

✓ *XML is a meta-language for markup: it does not have a fixed set of tags but allows users to define tags of their own.*

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML.

### 2.1 Introduction

#### 2.1 Introduction

#### 2.2 The XML language

#### 2.3 Structuring

#### 2.4 Namespaces

#### 2.5 Addressing and Querying XML documents

#### 2.6 Processing

#### 2.7 Summary

#### Suggested Reading.

#### Exercises and Projects

### ■ XML is an extensible language

- ✓ Communities and business sectors are in process of defining their specialized vocabularies, creating XML applications (or extensions of XML).
- ✓ Such applications have been defined in many domains: maths (MathML), bioinformatics (BSML), human resources (HRML), astronomy (AML), news (NewsML) and investment (IRML).
- ✓ Also, the W3C has defined many languages on top of XML, such as SVG, SMIL and RDF (see chapter 3), etc.
- ✓ XML can serve as a uniform data exchange format between applications

## 2 Structured Web Documents in XML. 2.2 The XML Language

- An XML document consists of a prolog, a number of elements and an epilog (optional, not discussed here)

### 2.2.1 Prolog

- ✓ The prolog is an XML declaration and an optional reference to external structuring documents.

Example: `<?xml version="1.0" encoding="UTF-16"?>`

- ✓ It specifies that the current document is an XML docu and defines the version and the character encoding used in the particular system (UTF-8, UTF-16 and ISO 8859-1). The character encoding is not mandatory, but it is a good practice.

- ✓ Sometimes we also specify whether the document is self-contained, that is, whether it does not refer to external structuring documents.

`<?xml version="1.0" encoding="UTF-16" standalone="no"?>`

- ✓ A reference to external structuring documents looks like this:

`<!DOCTYPE book SYSTEM "book.dtd"?>`

Where the structuring inf is found in a local file called **book.dtd**.

- ✓ The reference may be also a **URL**. If only a locally recognized name or only a URL is used, then the label **SYSTEM** is used. However, if one wishes to give both a local name and a URL, then the label **PUBLIC** should be used instead.

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.2 The XML Language

- An XML document: a *prolog*, a number of *elements* and an *epilog* (optional)

### 2.2.2 Elements

✓ XML elements represent the “things” the XML document talks about, such as: *books*, *authors* and *publishers*.

Example:      `<lecturer>Juan Perez</lecturer>`

✓ Tag names can be chosen almost freely; there are few restrictions. The main restriction are:

✓ 1. The first character must be: a letter, an underscore, or a colon;

✓ 2. no name may begin with the string “xml” in any combination cases (such as “Xml” and “XML”).

✓ The content may be a text, or other element, or nothing.

Example:      `<lecturer>`  
                  `<name>Juan Perez</name>`  
                  `<phone>+34-91-549-57-00</phone>`  
                  `</lecturer>`

✓ If there is no content, the element is called *empty*. Ex.: `<lecturer></lecturer>`  
and can be abbreviated as `<lecturer/>`

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.2 The XML Language

- An XML document: a *prolog*, a number of *elements* and an *epilog* (optional)

**2.2.3 Attributes.** An attribute is a name-value pair inside the opening tag of an element.

Example: `<lecturer name= "Juan Perez" phone= "+34-91-549-57-00"/>`

Example of attributes for a nonempty element:

```
<order orderNo= "23456" customer= "Juan Perez" date= "April 19, 2005">
  <item itemNo= "a528" quantity= "1"/>
  <item itemNo= "c817" quantity= "3"/>
</order>
```

The same inf could have been written replacing attributes by nested elements:

```
<order>
  <orderNo> 23456 </orderNo>
  <customer>Juan Perez</customer>
  <date>April 19, 2005</date>
  <item>
    <itemNo>a528</itemNo>
    <quantity>1</quantity>
  </item>
  <item>
    <itemNo>c817</itemNo>
    <quantity>3</quantity>
  </item>
</order>
```

When to use elements and when attributes is often a matter of taste. However, NOTE that *attributes cannot be nested.*

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.2 The XML Language

- An XML document: a *prolog*, a number of *elements* and an *epilog* (optional)

**2.2.4 Comments.** A comment is a piece of text to be ignored by the parser. It has the form

```
<!-- This is a comment -- >
```

**2.2.5 Processing Instructions (PIs).** PIs provide a mechanism for passing inf to an application about how to handle elements. The general form is:

```
<?target instruction ?>
```

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

PIs offer procedural possibilities in otherwise declarative environment.

**2.2.6 Well-Formed XML Documents.** An XML document is well-formed if it is syntactically correct, according to some rules:

- ✓ There is only one outermost element in the document, called the *root element*.
- ✓ Each element contains an opening and the corresponding closing tag
- ✓ tags may not overlap, as in `<author><name>Juan Perez</author></name>`
- ✓ Attributes within an element have unique names.
- ✓ Element and tag names must be permissible.

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.2 The XML Language

- An XML document: a *prolog*, a number of *elements* and an *epilog* (optional)

**2.2.7 The Tree Model of XML Documents.** Well-formed documents could be represented as trees (an instructive representation). Consider the following document:

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE email SYSTEM "email.dtd">
<email>
  <head>
    <from name="Juan Perez"
          address="juanperez@sinbad.dit.upm.es" / >
    <to name="Luis Martin"
        address="luismartin@sinbad.dit.upm.es" / >
    <subject>Dónde está el borrador?</subject>
  </head>
  <body>Luis, dónde está el borrador del artículo que me prometiste la semana pasada?
  </body>
</email>
```

Figure 2.1 shows the tree representation of this XML document. It is an ordered labelled tree:

- There is exactly one *root*.
- There are no cycles
- Each node, other than root, has exactly one parent
- The order of elements is important

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

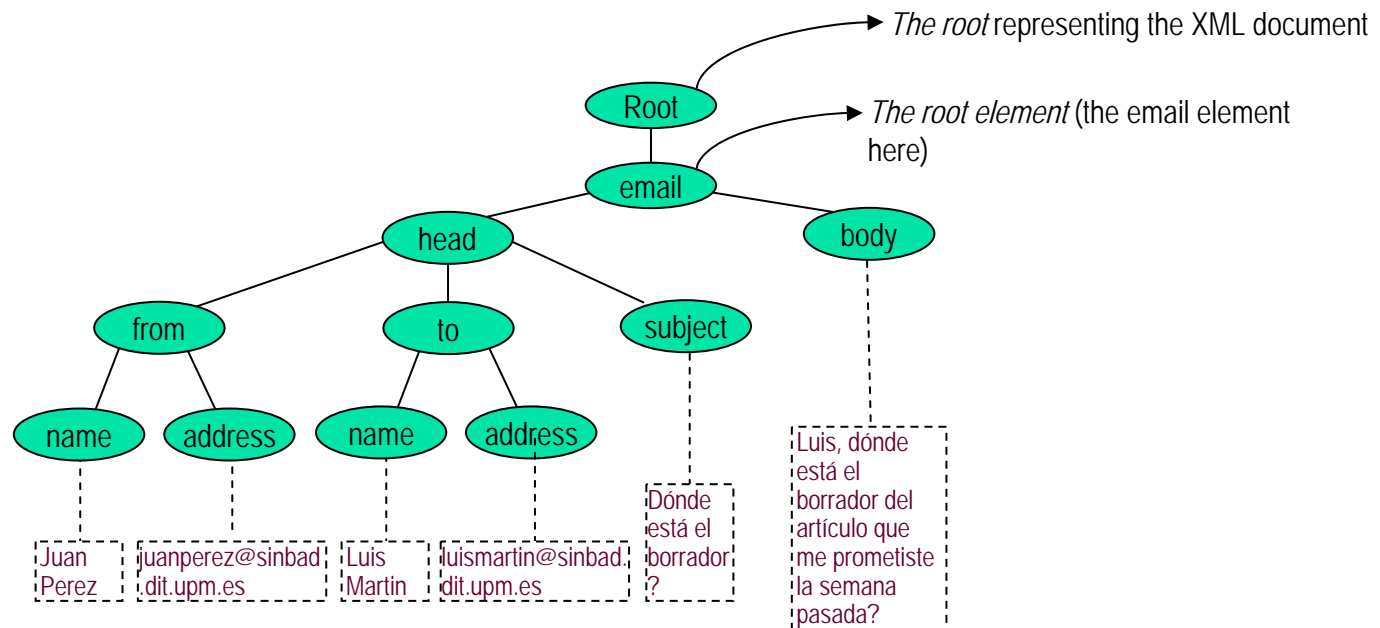
Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.2 The XML Language

- An XML document: a *prolog*, a number of *elements* and an *epilog* (optional)

**2.2.7** Figure 2.1: **Tree Representation of an XML document.** It is an ordered labeled tree: 1) There is exactly one root. 2) There are no cycles. 3) Each node, other than root, has exactly one parent.



4) The order of elements is important. However, the order of attributes is not important

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.3 Structuring

- An XML document: a *prolog*, a number of *elements* and an *epilog* (optional)

### 2.3 Structuring.

An XML doc is well-formed if it follows some syntactic rules. But these rules say nothing specific about the structure of the doc.

Suppose that 2 applications try to communicate, and they wish to use the same vocabulary.

For this purpose, it is necessary to define all the element and attribute names that may be used. Moreover, the structure should also be defined: what values an attribute may take, which elements must occur within other elements, and so on.

*An XML doc is valid* if it is well-formed, uses structuring inf and respects that structuring information.

There are two ways of defining the structure of XML docs: *DTDs* (the older and more restricted way) and *XML Schema* which offers extended possibilities, mainly for the definition of data types.

- 2.1 Introduction
- 2.2 The XML language
- 2.3 Structuring
- 2.4 Namespaces
- 2.5 Addressing and Querying XML documents
- 2.6 Processing
- 2.7 Summary
- Suggested Reading.
- Exercises and Projects

## 2 Structured Web Documents in XML. 2.3 Structuring

- An XML document: a *prolog*, a number of *elements* and an *epilog* (optional)

### 2.3.1 DTDs.

**External DTD** if the component of a DTD is defined in a separate file (the better option)

**Internal DTD** if it is defined within the XML doc itself (no reusability, consistency is more hard to maintain among many copies of the same doc etc.)

**Elements.** Consider the element

```
<lecturer>
  <name>Juan Perez</name>
  <phone>+34-91-549-57-00</phone>
</lecturer>
```

A DTD for this element type is:

```
<! ELEMENT lecturer (name. phone) >
<! ELEMENT name (#PCDATA) >
<! ELEMENT phone (#PCDATA) >
```

Where the meaning is:

- ✓ The element types `lecturer`, `name`, `phone` may be used in the doc
- ✓ A `lecturer` element contains a `name` element and a `phone` element, in that order
- ✓ A `name` element and a `phone` element may have any content. In DTDs, `#PCDATA` is the only atomic type for elements.

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.3 Structuring

### 2.3.1 DTDs. Elements.

For expressing that a `lecturer` element contains a `name` element or a `phone` element as follows:

```
<! ELEMENT lecturer (name | phone) >
```

For expressing that a `lecturer` element contains a `name` element and a `phone` element in any order:

```
<! ELEMENT lecturer ((name, phone) | (phone, name)) >
```

**Attributes.** Consider now this element:

```
<order orderNo= "23456" customer= "Juan Perez" date= "April 19, 2005">  
  <item itemNo= "a528" quantity= "1"/>  
  <item itemNo= "c817" quantity= "3"/>  
</order>
```

A DTD for it looks like this:

```
<! ELEMENT order (item+)>  
<! ATTLIST order  
  orderNo          ID          #REQUIRED  
  customer  CDATA    #REQUIRED  
  date            CDATA    #REQUIRED>  
<! ELEMENT item EMPTY>  
<! ATTLIST item  
  itemNo          ID          #REQUIRED  
  quantity        CDATA    #REQUIRED  
  comments  CDATA    #IMPLIED>
```

New aspects: the `item` element type *is defined to be empty* and `item+` that is the cardinality operators (`?`: appears zero times or once; `*`: appears zero or more times; `+`: appears one or more times). No cardinality operator means exactly one.

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.3 Structuring

### 2.3.1 DTDs.

**Attributes.** an XML element definition can also define attributes in an *attribute list*.

```
<! ATTLIST order
      orderNo      ID          #REQUIRED
      customer     CDATA      #REQUIRED
      date         CDATA      #REQUIRED>
```

The first component is the name of the element type to which the list applies, followed by a list of triplets of: *attribute name, attribute type, value type*

An attribute name is a name that can be used in an XML doc using a DTD.

**Attribute Types.** They are similar to data types, but very limited:

- **CDATA**, a string (sequence of characters)
- **ID**, a unique name across the entire XML doc
- **IDREF**, a reference to another element with an ID attribute carrying the same value as the IDREF attribute
- **IDREFS**, a series of IDREFs
- $(v_1 | \dots | v_n)$ , an enumeration of all possible values

**Value Types.** They are four value types:

- **#REQUIRED**, the att must appear in every occurrence of the element type in the XML doc.
- **#IMPLIED**, the appearance of the att is optional
- **#FIXED "value"**, every element must have this att, which has always the given after #FIXED in the DTD. A value given in an XML doc is meaningless because it is overridden by the fixed value.
- **"value"**, This specifies the default value for the att

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.3 Structuring

### 2.3.1 DTDs. Referencing. An example for the use of IDREF and IDREFS. Before, we give a DTD:

```
<! ELEMENT family (person*)>
<! ELEMENT person (name)>
<! ELEMENT name (#PCDATA)>
<! ATTLIST person
    id      ID          #REQUIRED
          mother      IDREF   #IMPLIED
          father      IDREF   #IMPLIED
          children    IDREFS  #IMPLIED>
```

An XML element that respects this DTD is the following:

```
<family>
  <person id="Juan" mother="Maria" father="Pedro">
    <name>Juan Perez</name>
  </person>
  <person id="Bridget" mother="Maria">
    <name>Bridget Jones</name>
  </person>
  <person id="Mary" children="Juan Bridget">
    <name>Mary Poppins</name>
  </person>
  <person id="Pedro" children="Juan">
    <name>Pedro Perez</name>
  </person>
</family>
```

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.3 Structuring

### 2.3.1 DTDs. A Concluding Example: A DTD for the previous email element.

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

```
<! ELEMENT email (head, body)>
<! ELEMENT head (from, to+, cc*, subject)>
<! ELEMENT from EMPTY>
<! ATTLIST from
    name CDATA #IMPLIED
    address CDATA #REQUIRED>
<! ELEMENT to EMPTY>
<! ATTLIST to
    name CDATA #IMPLIED
    address CDATA #REQUIRED>
<! ELEMENT cc EMPTY>
<! ATTLIST cc
    name CDATA #IMPLIED
    address CDATA #REQUIRED>
<! ELEMENT subject (#PCDATA)>
<! ELEMENT body (text, attachment*)>
<! ELEMENT text (#PCDATA)>
<! ELEMENT attachment EMPTY>
<! ATTLIST attachment
    encoding (mime |binhex) "mime"
    file CDATA #REQUIRED>
```

## 2 Structured Web Documents in XML. 2.3 Structuring

### 2.3.1 DTDs. A Concluding Example: A DTD for the previous `email` element.

#### Remarks. :

- A **head** element contains a **from** element, at least one **to** element, zero or more **cc** elements and a **subject** element, in that order.
- In **from**, **to** and **cc** elements the `name` att is not required; the **address** att, on the other hand, is always required
- A **body** element contains a **text** element, possibly followed by a number of **attachment** elements
- The **encoding** att of an **attachment** must have either the value `"mime"` or `"binhex"`, the former being the default value

#### Two remarks on DTDs:

- a) A DTD can be interpreted as an Extended Backus-Naur Form (EBCNF). For example, the declaration `<! ELEMENT email (head, body)>`  
Is equivalent to the rule `email ::= head body` expressing that an e-mail consists of a head followed by a body.
- b) Recursive definitions are possible in DTD. For example :  
`<! ELEMENT bintree ((bintree root bintree) | emptytree)>`  
Defines binary trees: a binary tree is the empty tree, or consist of a left sub-tree, a root and a right sub-tree.

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.3 Structuring

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

**2.3.2 XML Schema:** XML Schema offers a richer language for defining the structure of XML docs than DTDs.

- Its syntax is based on XML itself, what provides a significant improvement in readability, and, the most important:
- *it allows significant reuse of technology and refining schemas.*
- It allows to define new types by extending or restricting already existing ones
- Finally, XML Schema provides a sophisticated set of data types that can be used in XML docs (DTDs were limited to string only)

**An XML schema is an element with an opening tag like:**

`<xsd: schema`

```
xmlns : xsd= "http://www.w3.org/2000/10/XMLSchema" version="1.0">
```

The element uses the schema of XML Schema, found at the W3C web site. It is the foundation on which new schemas can be built.

The prefix **xsd** denotes the namespace of that schema (later discussed). If the prefix is omitted in the **xmlns** attribute, then we are using elements from this namespace by default:

`<schema`

```
xmlns="http://www.w3.org/2000/10/XMLSchema" version="1.0">
```

In the following we omit the **xsd** prefix.

## Tabla: XML namespace prefixes y sus URIs

XML namespace prefix	XML namespace URI
xsd	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>
xsi	<a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a>
xdt	<a href="http://www.w3.org/2003/11/xpath-datatypes">http://www.w3.org/2003/11/xpath-datatypes</a>
sqlxml	<a href="http://standards.iso.org/iso/9075/2003/sqlxml">http://standards.iso.org/iso/9075/2003/sqlxml</a>

## 2 Structured Web Documents in XML. 2.3 Structuring

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

**2.3.2 XML Schema.** The most important contents of an XML Schema are the *definitions of element* and *attribute types* which are defining using data types.

**Element Types:** The syntax of element types is `<element name= "..."/>`

And they may have a number of optional attributes, such as types,

- `type= "..."` (later discussed), or cardinality constraints
- `minOccurs= "x"`, where **x** may be any natural number (including zero)
- `maxOccurs= "x"`, where **x** may be any natural number (including zero) or unbounded

`minOccurs` and `maxOccurs` are generalizations of the cardinality operators offered by DTDs (`?`, `*`, `+`). They have a default value equal to 1.

Examples:

```
<element name="email"/>
```

```
<element name="head" minOccurs="1" maxOccurs="1"/>
```

```
<element name="to" minOccurs="1"/>
```

## 2 Structured Web Documents in XML. 2.3 Structuring

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

**2.3.2 XML Schema.** The most important contents of an XML Schema are the *definitions of element* and *attribute types* which are defining using data types.

**Attribute Types:** The syntax of attribute types is `<attribute name= "..."/>`

And they may have a number of optional attributes, such as types,

- `type= "..."` , or existence (corresponds to **#OPTIONAL** and **#IMPLIED** in DTDs)
- `use= "x"` , where **x** may be **optional** or **required** or a **default** value (corresponds to **#FIXED** and default values in DTDs)
- `use= "x" value= "..."` , where **x** may be **default** or **fixed**

Examples:

```
<attribute name="id" type="ID" use="required"/>
```

```
<element name="speaks" type="Language" use="default" value= "en"/>
```

## 2 Structured Web Documents in XML. 2.3 Structuring

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

**2.3.2 XML Schema.** The most important contents of an XML Schema are the *definitions of element* and *attribute types* which are defining using data types.

**Data Types:** XML Schema provides powerful capabilities for defining data type. There is a variety of built-in data types. Some of them are the following:

- Numerical data types, including **integer, Short, Byte, Long, Float, Decimal.**
- String data types, including **string, ID, IDREF, CDATA, Language**
- Date and time data types, including, **date, time, Month, Year**

There are also *user-defined data types*, comprising *simple data types*, which can not use elements or attributes, and *complex data types*, which can use elements and attributes. Complex types are defining from existing data types by defining some attributes if any) and using

- **sequence**, a sequence of existing data type elements, the appearance of which in a predefined order is important
- **all**, a collection of elements that must be appear, but the order of which is not important
- **choice**, a collection of elements, of which one will be chosen

## 2 Structured Web Documents in XML. 2.3 Structuring

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

### 2.3.2 XML Schema. Data Types: An example of a Complex Data Type:

```
<complexType name="lecturerType">
  <sequence>
    <element name="firstname" type="string"
              minOccurs="0" maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
  </sequence>
  <attribute name="title" type="string" use="optional"/>
</complexType>
```

The meaning is that an element in an XML doc that is declared to be of type **lecturerType** may have a **title** attribute; it may also include any number of **firstname** elements and must include exactly one **lastname** element

### 2.3.2 XML Schema. Data Type Extension

### 2.3.2 XML Schema. Data Type Restriction

## 2 Structured Web Documents in XML. 2.4 Namespaces

**2.4 Namespaces:** XML, a universal (meta) markup lang, may access to inf from various sources.

So, technically speaking, an XML doc may use more than one DTD or schema. However, since each structuring doc was developed independently, *name clashes* appear inevitable.

If DTD *A* and DTD *B* define an element type *e* in different ways, a parser that tries to validate an XML doc in which *e* element appears must told which DTD to use for validating purposes. The technical solution is simple: using a different prefix for each DTD or schema.

The prefix is separated from the local name by a colon: **prefix : name**

Ex.: **"lecturers"** in USA (uky) are not consider regular faculty; however in Australian (gu) they correspond to Assistant Professor in USA. Lets to deal with this disambiguation

```
<?xml version="1.0" encoding="UTF-16"?>
<vu:instructors
  xmlns:vu="http://www.vu.com/empDTD"
  xmlns:gu="http://gu.au/empDTD"
  xmlns:uky="http://www.uky.edu/empDTD">
  <uky:faculty
    uky:title="assistant professor"
    uky:name="Juan Perez"
    uky:department="Computer Science"/>
  <gu:academicStaff
    gu:title="lecturer"
    gu:name="Luis Martin"
    gu:school="Information Technology"/>
</vu:instructor>
```

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.4 Namespaces

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

**Namespaces** are declared within an element and can be used in that element and any of its children (elements and attributes).

namespace declaration: **xmlns: prefix="location"**

Where **location** is the address of the DTD or schema.

If a prefix is not specified, as in **xmlns = "location"**, then the location is used by default. For example, the example in previous page is equivalent to the following doc:

```
<?xml version="1.0" encoding="UTF-16"?>
```

```
<vu: instructors
```

```
  xmlns: vu="http://www.vu.com/empDTD"
```

```
  xmlns= http://gu.au/empDTD
```

(gu: does not appear now)

```
  xmlns: uky="http://www.uky.edu/empDTD">
```

```
<uky: faculty
```

```
  uky: title="assistant professor"
```

```
  uky: name="Juan Perez"
```

```
  uky:department="Computer Science"/>
```

```
<academicStaff
```

```
  title="lecturer"
```

```
  name="Luis Martin"
```

```
  school="Information Technology"/>
```

(gu: does not appear here)

```
</vu:instructor>
```

## 2 Structured Web Documents in XML. 2.5 Addressing and Querying XML Docs

- 2.1 Introduction
- 2.2 The XML language
- 2.3 Structuring
- 2.4 Namespaces
- 2.5 Addressing and Querying XML documents
- 2.6 Processing
- 2.7 Summary
- Suggested Reading.
- Exercises and Projects

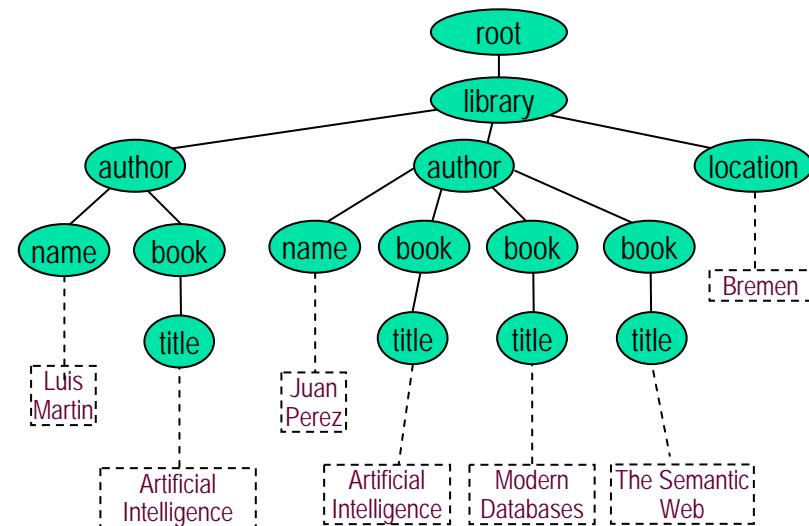
**Addressing and Querying XML Docs.** As SQL in RDB, XML docs have a number of proposals for QL, such as XQL, XML-QL and XQuery. The central concept of XML QL is a *path expression* node (or a set of nodes) in the tree representation of the XML doc can be reached.

*XPath* is a language for addressing parts of an XML doc. It operates on the tree data model of XML and has a non-XML Syntax. The key concepts are *path expression*. They can be:

- **Absolute** (starting at the root of the tree). Syntactically they begin with the symbol */*, which refers to the root of the doc, situated one level above the root element of the doc;
- **Relative** to a context node.

Consider the following doc:

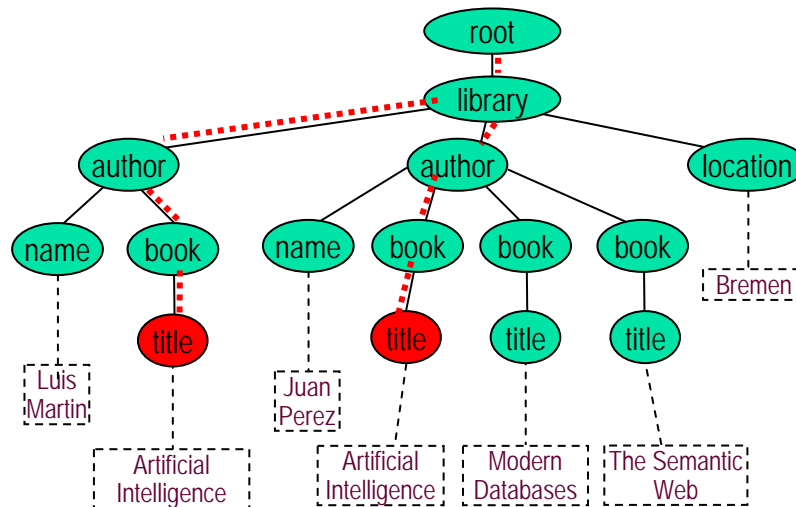
```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE library PUBLIC "library.dtd">
<library location="Bremen">
  <author name="Juan Perez">
    <book title="Artificial Intelligence"/>
    <book title="Modern Databases"/>
    <book title="The Semantic Web"/>
  </author>
  <author name="Luis Martin">
    <book title="Artificial Intelligence"/>
  </author>
</library>
```



## 2 Structured Web Documents in XML. 2.5 Addressing and Querying XML Docs

The capabilities of *XPath* are described by means of some examples of *path expression*.

1. Address all **author** elements: `//library/author`  
This *path expression* addresses all **author** elements that are children of the **library** node, which resides immediately below the root. Using a sequence  $/t_1/.../t_n$ , where each  $t_{i+1}$  is a child node of  $t_i$ , we define a path through the tree representation.  
An alternative solution for the previous example is `//author`. Here `//` says that we should consider all elements in the doc and check whether they are of type **author**. .....
2. Address the **location** attribute nodes within **library** element nodes `//library/@location`, where `@` denotes **attribute nodes**.
3. Address all **title** attribute nodes within **book** elements anywhere in the doc, which have the value "Artificial Intelligence"  
`//book / @ title = "Artificial Intelligence"`



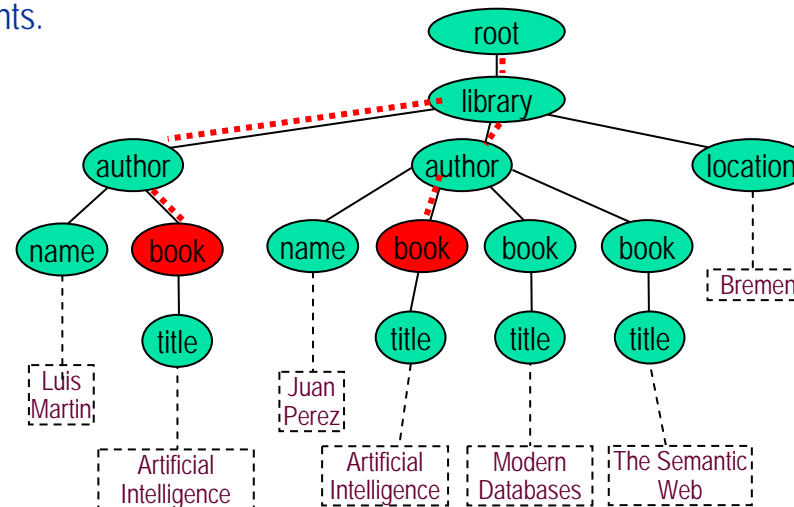
- 2.1 Introduction
- 2.2 The XML language
- 2.3 Structuring
- 2.4 Namespaces
- 2.5 Addressing and Querying XML documents
- 2.6 Processing
- 2.7 Summary
- Suggested Reading.
- Exercises and Projects

## 2 Structured Web Documents in XML. 2.5 Addressing and Querying XML Docs

The capabilities of *XPath* are described by means of some examples of *path expression*.

1. Address all **author** elements: `//library/author //author`.
2. Address the **location** attribute nodes within **library** element nodes `//library/@location`, where `@` denotes attribute nodes.
3. Address all **title** attribute nodes within **book** elements anywhere in the doc, which have the value "Artificial Intelligence"  
`//book/@title="Artificial Intelligence"`
4. Address all **book** with title "Artificial Intelligence"

`//book [ @ title="Artificial Intelligence"]`, where [...] is called *predicate* or *a filter expression*. It restricts the set of addressed nodes. Here, **book** is the element targeted, however in expression 3 we collected **title** attribute nodes of **book** elements.



- 2.1 Introduction
- 2.2 The XML language
- 2.3 Structuring
- 2.4 Namespaces
- 2.5 Addressing and Querying XML documents
- 2.6 Processing
- 2.7 Summary
- Suggested Reading.
- Exercises and Projects

## 2 Structured Web Documents in XML. 2.5 Addressing and Querying XML Docs

- 2.1 Introduction
- 2.2 The XML language
- 2.3 Structuring
- 2.4 Namespaces
- 2.5 Addressing and Querying XML documents
- 2.6 Processing
- 2.7 Summary
- Suggested Reading.
- Exercises and Projects

The capabilities of *XPath*: more examples of *path expression*.

5. Address the first **author** element node in the XML document: `//author [1]`.
6. Address the last **book** element within the first **author** element node in the doc  
`//author [1]/book [last()]`
7. Address all **book** element nodes without a **title** attribute  
`//book[not @title]`

*XPath* has a complicated full syntax. In general, a *path expression* consists of a series of steps, separated by slashes. A *step* consists of an *axis specifier*, a *node test* and an *optional predicate*.

An *axis specifier* determinates the tree relationship between the nodes to be addressed and the context node. Examples are: parent ancestor, child (the default), sibling, attribute node. Where `//` is such an axis specifier; it denotes descendent or self.

A *node test* specifies which nodes to address. The most common node tests are element names( which may use namespace information), but there are others. For example, `*` addressed all element nodes, `comment ()` all comment nodes, and so on.

*Predicates* (or *filter expressions*) are optional and used to refine the set of addressed nodes. For example, the exp `[1]` selects the first node, `[position()=last()]` selects the last node, `[position() mod 2 = 0]` the even nodes, and so on.

## 2 Structured Web Documents in XML. 2.6 Processing

Now we deal with how XML docs can be displayed. Unlike HTML docs, XML docs do not contain formatting inf. So, a given doc can be displayed in many ways, when different *style sheets* are applied to it.

Example: `<author>`  
`<name> Juan Perez </name>`  
`<affiliation> University of Bremen</affiliation>`  
`<email> jp@tzi.de </email>`  
`</author>`

Using a style sheet, the output could be : **Juan Perez**  
**University of Bremen**  
**jp@tzi.de**

But, through another style sheet, the output could be different, as:

*Juan Perez*  
University of Bremen  
jp@tzi.de

*Style sheets* can be written in different languages:

*CSS2* (Cascade Style Sheets level 2),

*XSL* (extensible stylesheet language) that includes both transformation lang (*XSLT*) and a formatting lang. Each of these is, of course, an XML application.

(no more details..., interested: see the w3c standard specification)

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

## 2 Structured Web Documents in XML. 2.7 Summary

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects

- XML is a meta-language that allows users to define markup for their *docs* using tags.
- Nesting of tags introduces structure. The structure of docs can be enforced using schemas or DTDs
- XML separates content and structure from formatting
- XML is the facto standard for representation of structured information on the Web and supports machine processing of information.
- XML supports the exchange of structured inf across different applications through markup, structure and transformations.
- XML is supported by query languages.

Some points discussed in subsequent chapters include:

- The nesting of tags does not have standard meaning
- The semantics of XML docs is not accessible to machines, only to people.
- Collaboration and exchange are supported if there is an underlying shared understanding of the vocabulary. XML is well-suited for close collaboration, where domain –or community-based vocabularies are used.
- XML is not well-suited for a global communication.

# Suggested Reading

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

**Suggested Reading.**

Exercises and Projects

Some books covering XML industrial technicalities in depth are:

- [Haro01], E.R. Harold, *XML Bible*, 2nd. edition, New York, Wiley (Hundry Minds), 2001
- [Merce01], D. Mercer, *XML: A Beginner's Guide*, New York, McGraw Hill (Osborne), 2001

Recent trends in XML querying may be found in:

- [<http://www.w3.org/XML/Query.html>](http://www.w3.org/XML/Query.html)

Other Web sites with teaching material on XML and related technologies:

- <http://www.xml.com>
- <http://www.topxml.com>
- <http://www.xslt.com>

## Exercises and Projects

### Ejercicios:

1. En el ejemplo de e-mail hemos especificado el cuerpo de un e-mail para que contenga exactamente un texto y un número de documentos adjuntos (attachments). Modificar el esquema para permitir que haya un número arbitrario de textos y de documentos adjuntos en cualquier orden.
2. Explorar en la Web aplicaciones XML conteniendo las palabras clave: "XML DTD" o "XML schema".

### Breves proyectos académicos:

1. Diseñar un vocabulario para modelar (parte de) su lugar de trabajo. Por ejemplo, en su universidad, diseñar el vocabulario sobre cursos, platilla de profesores, aulas, publicaciones, etc.
2. ¿Le gusta cocinar?. Diseñe un vocabulario sobre comidas, gustos y recetas.
3. ¿Es un inversor?. Diseñe un vocabulario sobre las posibles opciones de inversión. Por ejemplo, riesgos, retornos, edad del financiador, tipo financiador, etc.
4. Para alguno de sus *hobbies*, diseñe un vocabulario para intercambiar información con otros que comparten su afición.
5. ¿Lee libros de ciertas categorías?. Diseñe un vocabulario para describirlos e intercambiar información con otras personas.

2.1 Introduction

2.2 The XML language

2.3 Structuring

2.4 Namespaces

2.5 Addressing and Querying  
XML documents

2.6 Processing

2.7 Summary

Suggested Reading.

Exercises and Projects