



XML Applications

Prof. **Andrea Omicini**

DEIS, Ingegneria Due

Alma Mater Studiorum, Università di Bologna a Cesena

andrea.omicini@unibo.it

Outline

— [XHTML

— [XML Schema

— [XSL & XSLT

XHTML

HTML vs. XML

HTML

- Presentation oriented
- No structure, no semantics for data

XML

- Data oriented
- Allows for structural / semantic representation
- Can be validated through grammars

XHTML: An XML-based HTML

— [The idea: use XML rather than SGML to define an HTML equivalent
— so, XHTML is an XML application
— keeping most HTML tags with their original semantics
— but!
— with the properties of well-formedness and validity of XML

— [In fact, most browsers have extended support from HTML to XHTML
soon and easily

— <http://www.w3.org/MarkUp/2004/xhtml-faq>

— [Standard W3C

"The Extensible HyperText Markup Language (XHTML™) is a family of current and future document types and modules that reproduce, subset, and extend HTML, reformulated in XML"

— XHTML 1.0, 1.1, 2.0, Basic, etc.

Main differences

— [So, XHTML adds to HTML the same XML main rules

— perfect match between start and end tags

— no overlapping elements

— one and only one root elements

— attribute values are always quoted

— at most one attribute with a given name per element

— neither comments nor processing instructions within tags

— no unescaped > or & signs in the character data of elements or attributes

—

— ..

— [which were typical sources of problems in HTML

— [Plus, it adds case-sensitivity

— and all XHTML tags are lower-case

An XHTML Fragment

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>A0 Biographic Notes</title>
    <link href="style.css" rel="stylesheet" type="text/css" media="screen" />
    <script type="text/javascript" src="common.js"></script>
  </head>
  <body class="papers">
    <h1 class="header">Biographic Notes</h1>

    <div class="body">
      ..
    </div>

  </body>
</html>
```

XML Schema

Limitations of DTDs

- [DTDs are great but

- DTDs have no support for types
- DTDs have no way to define the element's content
- DTDs have SGML syntax
 - no XML syntax
 - no way to use XML technology for DTDs
 - e.g., no re-use of parsers
- DTDs have some limitations in expressiveness
 - e.g., sequences constrain child types as well as order
- DTDs have no support for namespaces

- [Why not use extensibility and flexibility of XML to define XML syntax?

- using XML as a meta-markup language to define a new XML application?

Goals of XML Schemas

- [Defining an XML application for XML validation

- [Supporting everything from DTDs, plus

- types

- in particular for element contents

- namespaces

- [Promoting re-use of all XML-related

- technologies

- like, say, XML parsers

- knowledge

- like, say, an human designer skilled at XML handling

Elements of XML Schemas: Pre-defined Simple Type Elements

— [For a type system to be supported, first some **pre-defined** types should be provided

— string, boolean, float, double, integer

— date

— binary

— uriReference

— pattern

— [Then, you can define your own simple types

Elements of XML Schemas: Simple Type Elements

- [`xsd:simpleType`

- [**Example**

```
<xsd:simpleType name="natural">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="0" />  
  </xsd:restriction>  
</xsd:simpleType>
```

- defines type `natural` as a restriction of integers to natural numbers

- [**Other keywords available**

- see specification

Elements of XML Schemas: Complex Type Elements

- [xsd:complexType

- [**Example**

```
<xsd:complexType name="complex">  
  <xsd:sequence>  
    <xsd:element name="real" type="xsd:float">  
    <xsd:element name="imaginary" type="xsd:float">  
  </xsd:sequence>  
</xsd:complexType >
```

- defines type `complex` as a pairing of real numbers

- [**Using element declarations...**

- most of the facets for simple types can be used as attributes for elements
- e.g., `minInclusive`,...

Elements of XML Schemas: Element Declarations

- [xsd:element

- [**Examples**

- <xsd:element name="point" type="complex">
 - <xsd:element name="goals" type="natural">

- [**Element declaration associates types to elements**

- from pre-defined, simple to complex types

- [**Element declarations make a given element admissible within the doc**

- again, what is not specified is not allowed

- [**What is missing now are attribute declarations...**

Elements of XML Schemas: Attribute Declarations

— [xsd:attribute

— [Example

```
<xsd:attribute name="team" type="string">  
<xsd:attribute name="team" type="boolean" use="required" default="false">
```

— [All attributes are declared as simple types

— [Only complex elements can have attributes

— [Attribute declarations make a given attribute admissible for an element of a given complex type within the doc

Elements of XML Schemas: Last Few Things

```
<xsd:schema xmlns:xsd="http://www.w3c.org/2001/XMLSchema">
```

— [Associates the XML Schema namespace to the `xsd` prefix

— Just after the XML Declaration

— since and XML Schema is first of all an XML document

```
<xsd:complexType mixed="true">
```

— [Complex Types are allowed to specify Mixed Content

— for mixed-content, narrative-oriented XML documents

XSL & XSLT

XSL: eXtensible Stylesheet Language

- XML-based stylesheet language

- <http://www.w3.org/Style/XSL/>

- XSL is a family of recommendations for defining XML document transformation and presentation

- XSL Transformations (**XSLT**)

- <http://www.w3.org/TR/xslt>

- language for transforming XML

- XML Path Language (**XPath**)

- <http://www.w3.org/TR/xpath>

- expression language used by XSLT to access or refer to parts of an XML document

- XSL Formatting Objects (**XSL-FO**)

- <http://www.w3.org/TR/xsl/>

- XML vocabulary for specifying formatting semantics

XSL Transformations

XSLT is a language for transforming the structure of an XML document

Why Transform XML?

- two main issues for XML
 - data separation from presentation
 - portability / transmission of information
- often, the two together

In any case, this means that XML documents are typically NOT used in the same form they come in

- thus, the need to transform XML documents

Also, DOM and SAX allow for XML transformation

- they are similar, and also procedural
 - a more high-level, declarative form should be possible
 - which is where XSLT comes in

An Example: Hello World, XML

— [helloworld.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>  
<?xml-stylesheet type="text/xsl" href="helloworld.xsl"?>  
<greeting>Hello, World!!</greeting>
```

— [works as the *input* for transformation

An Example: Hello World, HTML

— [helloworld.html

```
<html>
  <head>
    <title>Today's Greeting</title>
  </head>
  <body>
    <p>Hello, World!!</p>
  </body>
</html>
```

— [works as the (desired) *output* of transformation

An Example: Hello World, XSLT

helloworld.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method='html' version='1.0' encoding='iso-8859-1' indent='yes' />

<xsl:template match="/">
  <html>
    <head>
      <title>Today's Greeting</title>
    </head>
    <body>
      <p><xsl:value-of select="greeting" /></p>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

actually *transforms* the XML input into the desired HTML output

Experiments

— { Browsers

— { A meta-processor for XSLT

XSLT in Short

Transformation rules are expressed through **templates**

- every template indicates which *parts* of the XML documents it matches with
 - through an **XPath expression** in its specification
- template is activated for all and only the tree nodes of the XML document that match the XPath expression
 - if more than one template match with the same expression, the template to apply is chosen non-deterministically
 - unless import or priorities are of concern
- always a root template activating the other templates
 - matching with the "root" expression "/"
 - if only one template, no need to specify the template element
- templates can activate each other recursively through the recursive rule `<xsl:apply-templates/>`

Just a matter to understand the mechanism and the syntax

Another Example of a XSLT sheet

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="emphasis">
  <i><xsl:apply-templates/></i>
</xsl:template>

</xsl:stylesheet>
```

— [transforms

```
<?xml version='1.0'?>
<para>This is a <emphasis>test</emphasis>.</para>
```

— [into

```
<?xml version="1.0" encoding="utf-8"?>
<p>This is a <i>test</i>.</p>
```

XSLT is Declarative

— [XSLT is a **declarative** language

— no side effects

— single assignement variables

— non-destructive assignment

— [This frees us from the burden of *how*

— leaving us only with the need for specifying *what*

Where to Use XSLT?

Data Conversion scenarios

- when there are

- different ways to represent the same things

- chunks of knowledge from different sources to be put together

- from XML to XML

- but also from anything to anything, just using the right parser / writer

Publishing scenarios

- typically meant to humans

- through a possibly huge range of different media and scenarios

- XML handles knowledge independently of the presentation

- but then presentation is often needed in the end

— And, the two things together, more often today

XPath

Expressions are part of the XSL specification

— defined as stand-alone component since they are used in other contexts, such as XLink & XPointer

Used throughout XSLT to select data from the source and manipulate it

Syntax defined through *production rules*

— like many grammars you already know, maybe

The language is complex and articulated

— better to learn by need, for you

Examples

— `chapter//footnote` selects all the child node `footnote` of node `chapter` which is child of the context node

— `attribute::colour` selects the `colour` attribute of the context node

XML Formatting Objects (XSL-FO)

- XML application to describe the layout of a page / presentation
 - a sort of page-description language à la PostScript, without a programming language

- XSL-FO provides a more sophisticated and flexible visual layout model than HTML + CSS

- like right-to-left and top-to-bottom text, footnotes, margin notes, page numbers in cross-references, etc.

- more or less generalises over HTML+CSS

- in fact, you may easily find the same property specification as CSS

- **56 elements**

- in the <http://www.w3.org/1999/XSL/Format> namespace

- rectangular **areas** with *formatting properties*

CSS vs. XSL

What to choose between CSS and XSL?

- CSS and XSL overlap to some extent

CSS advantages

- simple, specific, well supported by all browsers

XSL advantages

- more powerful, more general, goes far beyond mere presentation

So, even though they overlap a bit, they have different goals and scopes

- so they can live together for a while

- in the long run, XSL is the obvious front-runner

- but simplicity, support and legacy have often won over any other consideration