Università degli Studi di Bari

Laboratorio di Acquisizione della Conoscenza e Apprendimento delle Macchine

<?XML ?> Tutorial

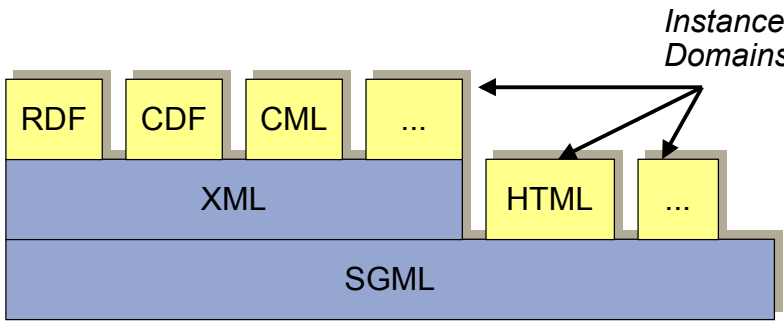
Pasquale Lops
Giovanni Semeraro
Dip. Informatica - Università di Bari

What is XML ?

Extensible Markup Language (XML) is a W3C proposed recommendation for a file format to easily and cheaply distribute electronic documents on the World Wide Web

XML is a meta-markup language that provides a format for describing structured data

Architectural Dependencies



Example Documents

- Books
- User manuals
- Product catalogs
- Order forms
- Medical documents
- Tax forms
- Mathematical formulas
- Chemical formulas
- Drug descriptions
- Dictionaries
- Newspapers
- Style sheets
- Musical scores
- Library indices
- Protein sequences
- Bibliographies
- Database schemas
- …

Goals

- **XML ensures that structured data will be uniform and independent of applications or vendors.**
- **This resulting interoperability is kick-starting a new generation of business and electronic-commerce Web applications.**
- **XML provides a data standard that can encode the content, semantics, and schemata for a wide variety of cases ranging from simple to complex**
- **XML maintains the separation of the user interface from the structured data**

Structural Representation of Data

From a technical point of view, XML can be used to mark up the following:

- An ordinary document.
- A structured record, such as an appointment record or purchase order.
- An object with data and methods, such as the persistent form of a Java object or ActiveX control.
- A data record, such as the result set of a query.
- Meta-content about a Web site, such as Channel Definition Format (CDF).
- Graphical presentation, such as an application's user interface.
- Standard schema entities and types.
- All links between information and people on the Web.

XML Documents

- XML is a text-based format, similar to HTML
- An XML source is made up of XML elements, each of which consists of a start tag (<title>), an end tag (</title>), and the information between the two tags (referred to as the content).
- Unlike HTML, XML allows an unlimited set of tags, each indicating not how something should look, but what something means.

Example: E-Mail Document in XML

```
<?XML VERSION="1.0"?>
<!-- This is a sample email data file -->
<!DOCTYPE mail SYSTEM "email.dtd" [
        <!ENTITY ingo    "Ingo.Macherius@tu-clausthal.de" >
        <!ENTITY henning "hb@ix.heise.de"                 > ]>
<mail>
                <Recipient>&henning;</Recipient>
                <Sender>&ingo;</Sender>
                <Date>Mon, 21 Apr 1997 09:27:55 +0200</Date>
                <Subject>XML literature</Subject>
        <Textbody>
                <p>Hello Mr <Name>Behme</Name>,</p>
                <p>Please read <Name>Jon Bosak</Name>'s introductory</p>
                <p> text "SGML, Java and the Future of the Web"</p>
                <p>Best wishes,</p>
                <p><Name>Ingo Macherius</Name></p>
        </Textbody>
</mail>
```

XML Features in a Nutshell

- **Extensibility**
  - Unlimited set user-defined tags
  - Can be used in any domain
- **Structure**
  - Can represent trees and graph structures (database schemas, OO hierarchies, …)
- **Validation**
  - Consuming applications can check for structural validity on importation

Authoring guidelines

There are several key things to remember when constructing a basic XML document:

- All elements must have an end tag.
- All elements must be cleanly nested (overlapping elements are not allowed).
- All attribute values must be enclosed in quotation marks.
- Each document must have a unique first element, the root node.

Examples

```
<books>
  <book isbn="0345374827">
    <title>The Great Shark Hunt</title>
    <author>Hunter S. Thompson</author>
  </book>
</books>
```

```
<books>
  <book isbn=0345374827>
    <title>The Great Shark Hunt
    <author>Hunter S.Thompson</title></author>
  </book>
</books>
```


XML Markup Overview

XML documents contain

- Character Data
- Comments & Escaped content
- Processing instructions
- Elements
- Document Type Definition Markup

Character Data

- Unicode (ISO 10646) characters without markup
- Example:

```
<p>Hello Mr <Name>Behme</Name>,</p>
<p>Please read <Name>Jon Bosak</Name>'s introductory text</p>
<p>"SGML, Java and the Future of the Web"</p>
<p>Best wishes,</p>
<p><Name>Ingo Macherius</Name></p>
```


Comments & Escaped Content

- Ignored during processing
- Example:

```
<!-- This is a sample email data file -->
<![CDATA[   … any markup here … ]]>
```

Processing Instructions

- Special instructions to the XML consumer application
- Example:

```
<?XML VERSION="1.0" encoding="UTF-8"?>
```


Elements

- Consists of a *start tag*, *body*, and *end tag*
- *Body* can be any other markup (even nested)
- Examples:

```
<p>Best wishes,</p>
<p><Name>Ingo Macherius</Name></p>
<p></p>
```

- Empty element shorthand:

```
<p/>
```

Attributes

- Optional (Attribute, Value) pairs associated with elements
- Example:

```
<person firstname="John Bosak" surname="John">
   <address>Sun MicroSystems</address>
   <e-mail>bosak@sun.com</e-mail>
</person>
```


DocTypeDeclarations

- Identifies the XML instance being used with a reference to the *Document Type Definition* (DTD)
- Includes definitions of entities
- Example:

```
<!DOCTYPE mail SYSTEM "email.dtd" [
        <!ENTITY ingo    "Ingo.Macherius@tu-clausthal.de" >
        <!ENTITY henning "hb@ix.heise.de" >
]>
```

Internal Text Entities

Entities are similar to variables in programming languages

```
<!ENTITY email_dib "info@di.uniba.it" >
```

"For more information, send a message to &email_dib;"

"For more information Please, send a message to info@di.uniba.it"

Built-in entities

- < for '<'
- > for '>'
- & for '&'
- ' for ' ' '
- " for ' " '

< Sanford & son >

< Sanford & son >

External entities

Entity defined in a local file

```
<!ENTITY myentity SYSTEM "/ENTS/MYENTITY.XML" >
```

Entity defined in a remote file

```
<!ENTITY myentity PUBLIC "-//MyCorp//ENTITIES//" >
```


Document Type Definition (DTD)

- Identifies the syntax of the XML "flavor" being used, i.e. CDF, RDF, CML, ...
- Meta-information about the document contents:
  - Valid element names
  - Valid attribute names and values
  - How elements can nest in each other
- Typically the DTD is stored in a separate document
- DTD does not say anything about document semantics
- DTD is an optional feature of XML

Well-formed vs. Valid Documents

- **Well-formed**
  - Conforms to the basic XML syntax (see Authoring Guidelines)
  - Can be parsed without regard to the DTD
  - <?XML version="1.0" standalone="yes" encoding="UTF-8"?>
- **Valid**
  - Well-formed
  - Conforms to its DTD
  - <?XML version="1.0" standalone="no" encoding="UTF-8"?>

Writing a DTD

*** External DTD ***

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

*** Internal DTD ***

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>
```

DTDs are expressed in XML

DTD ElementDeclarations

- Specifies a valid element and its valid contents
  (What can be nested inside the element?)
- Uses regular expressions to define valid contents
- Examples:

```
<!ELEMENT br EMPTY>                  // empty element
<!ELEMENT p ANY>                     // allows everything
<!ELEMENT mail
        (subject | from | to | textbody)* )>
```


Modelgroups

A model group is used to describe enclosed elements and text

Sequence

```
<!ELEMENT elementName (a,b,c,d) >
```

Choice

```
<!ELEMENT elementName (a | b | c | d) >
```

Mixed

```
<!ELEMENT elementName (a , b , (c | d)) >
```

Quantity control

The DTD author can dictate how often an element can appear at each location

Required element (not repeatable)	`(title,author)`
Optional element (not repeatable)	`(title,author?)`
Required element (repeatable, at least one)	`(title,chapter+)`
Optional element (repeatable)	`(title,chapter*)`

A model group may itself have an occurrence indicator

ex.: (a , b)+ **or** (a , b)* **or** (a , b)?

PCData

The location where the text is allowed are indicated by the keyword 'PCDATA'

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

<subject>XML Tutorial</subject>
```

DTD AttributeListDeclarations

- Defines allowed attribute names and values of an element
- Example:

```
<!ATTLIST list
        listtype  (bullets|ordered|glossary)  "glossary"
        name      CDATA                       #REQUIRED
>
```

Default value

Name

Type

#REQUIRED

#IMPLIED

DTD : Some Attribute Types

- **CDATA** - Any value
- **ID** - Unique identifier for the XML Element
- **IDREF** - Reference to an element with a specific ID
- **IDREFS** - Sequence of IDREFs
- **NMTOKEN** -
- **NMTOKENS**
- **ENTITY**
- …

Quick review

- Comments & Escaped content
  `<!-- …>` `<![CDATA[ … ]]>`
- Processing instructions
  `<?XML … ?>`
- Elements
  `<A href="http://www.w3.org">W3C</A>`
- Document Type Definition Markup
  `DOCTYPE, ELEMENT, ATTLIST`

Namespaces

- **An XML namespace is a collection of names that can be used as element or attribute names in an XML document.**
- **The namespace qualifies element names uniquely on the Web in order to avoid conflicts between elements with the same name.**
- **The namespace is identified by some URI (Universal Resource Identifier), URIs are used simply because they are globally unique across the Internet.**
- **The namespace is used in order to define a domain**

Namespace declaration

```
<BOOKS>
  <bk:BOOK xmlns:bk="urn:BookLovers.org:BookInfo"
           xmlns:money="urn:Finance:Money"
           xmlns:people="urn:RegistryOffice:People" >
    <bk:TITLE>A Suitable Boy</bk:TITLE>
    <bk:PRICE money:currency="US Dollar"> 22.95
    </bk:PRICE>
    <bk:AUTHOR>
          <people:TITLE>Dr.</people:TITLE>
          <people:NAME> J.Smith </people:NAME>
    </bk:AUTHOR>
  </bk:BOOK>
</BOOKS>
```


XML Schema: What is?

A document that describes what a correct document may contain

Specifically, a W3C Recommendation for an XML-document syntax that describes the permissible contents of XML documents

Created by W3C XML Schema Working Group based on many different submissions

What's wrong with DTDs?

- Unusual, non-XML like syntax
- No data typing, especially for element content
- Limited extensibility
- Only marginally compatible with namespaces
- …

Why XML Schema? …

- Enhanced data types
  - 44+versus 10
  - Can create your own datatypes
- Written in the same syntax as instance documents
  - Less syntax to remember

… Why XML Schema?

- Can specify element content as being unique and uniqueness within a region
- Can define multiple elements with the same name but different content
- Can define elements with nil content

XML Schema: Structure

An XML Schema document is enclosed into the <*schema*> tag and it can contain:

- <*import*> and <include> to insert other Schema document
- <*simpleType*> and <*complexType*> to define data types
- <*element*> and <*attribute*> to define global elements and attributes of the document
- <*attributeGroup*> and <*group*> to define set of attributes and groups of complex content model
- <*annotation*> to insert comments

XML Schema: types...

- **Simple types: cannot have children or attributes (string, decimal float, boolean, uriReference, date, time, ecc.)**
- **Complex types: can have child elements and attributes**

... types (Example)

```
<xsi:simpleType name='bodytemp' base='decimal'>
    <xsi:precision value='3'/>
    <xsi:scale value='1'/>
    <xsi:minInclusive value='36.5'/>
    <xsi:maxInclusive value='44.0'/>
</xsi:simpleType>

<xsi:complexType name='name'>
    <xsi:element name='forename' minOccurs='0' maxOccurs='*'/>
    <xsi:element name='surname'/>
</xsi:complexType>
```

XML Schema: Facets

For each type I can define *facets*, that specify the constrains of the type:

- *length, minLength, maxLength:* number demanded, min and max of characters
- *minExclusive, minInclusive*, *maxExclusive*, *maxInclusive*: min and max value, inclusive and exclusive
- *pattern*: regular expression that value must soddisfy
- *enumeration*: set of values
- ecc.

XML Schema: deriving types...

New simple types are defined by deriving them from existing atomic types building a complex tree

It is possible to extend types in two ways:

- by restriction (*derivedBy*="*restriction*") i.e. specifying added facets.
- by extension (*derivedBy*="*extension*") i.e. extending the content model

… deriving types… (Example 1)

```
<xsi:simpleType name='bodytemp' base='decimal'>
    <xsi:precision value='3'/>
    <xsi:scale value='1'/>
    <xsi:minInclusive value='36.5'/>
    <xsi:maxInclusive value='44.0'/>
</xsi:simpleType>

<xsi:simpleType name='healthbodytemp' base='bodytemp' derivedBy="restriction">
    <xsi: maxInclusive value='37.0'/>
</xsi:simpleType>
```


… deriving types (Example 2)

```
<xsi:complexType name='name'>
    <xsi:element name='forename' minOccurs='0' maxOccurs='*'/>
    <xsi:element name='surname'/>
</xsi: complexType >

<xsi: complexType name='fullname' base="name" derivedBy="extension">
    <xsi: element name='title' minOccurs='0'/>
</xsi: complexType >
```

XML Schema: complex types

To define a complex type I have to specify a content model and a set of valid attributes, i.e.

- ***Name***: tag name or attribute name
- ***Ref***: element name or attribute name defined in the global area
- ***Type***: type name
- ***maxOccurs***, ***minOccurs***: min and max number of occurences
- ***Fixed***, ***default***, ***nullable***, etc.

XML Schema: attributes

The attribute is defined by the tag <*attribute*> inside the element definition

```
<!ELEMENT A (B,C)>
<!ATTLIST A        p CDATA #IMPLIED          q (uno|due|tre) #IMPLIED>
```

VS

```
<xsi:element name="A">
    <xsi:element name="B" />
    <xsi:element name="C" />
    <xsi:attribute name="p" type="xsi:string" />
    <xsi:attribute name="q">
          <xsi:simpleType base="xsi:string">
                <xsi:enumeration value="uno" />
                 <xsi:enumeration value="due" />
                 <xsi:enumeration value="tre" />
           </xsi:simpleType>
     </xsi:attribute>
</xsi:element>
```

XML Schema: groups of attributes

```
<xsi:element name="A">
    <xsi:element name="B" />
    <xsi:element name="C" />
    <xsi:attributeGroup ref="attrsA" />
</xsi:element>

<xsi: attributeGroup name="attrsA">
    <xsi:attribute name="p" type="xsi:string" />
    <xsi:attribute name="q">
            <xsi:simpleType base="xsi:string">
                    <xsi:enumeration value="uno" />
                     <xsi:enumeration value="due" />
                     <xsi:enumeration value="tre" />
             </xsi:simpleType>
     </xsi:attribute>
</xsi: attributeGroup >
```


XML Schema: annotations

It is possible to insert comments in the schema document by the tag <*annotation*>

XML Schema provides three elements for annotating schemas

- <*annotation*> describes the schema
- <*documentation*> describes the schema for human readers
- <*appInfo*> describes the schema for computer programs

```
<xsi:element name='pippo'>
    <annotation>
            <documentation>pippo element</documentation>
    </annotation>
    <xsi:element name="pluto" />
</xsi:element>
```

Linking to the XML Schema

With the <*schemaLocation*> tag into the XML document we can link to the XML Schema document

```
<fv:pippo xmlns:fv="http://www.pippo.org/Pippo
    xmlns:xsi="http://www.w3.org/XMLSchema/istance"
    xsi:schemaLocation="http://www.pippo.org/pippo.xsi">
----
</fv:pippo>
```

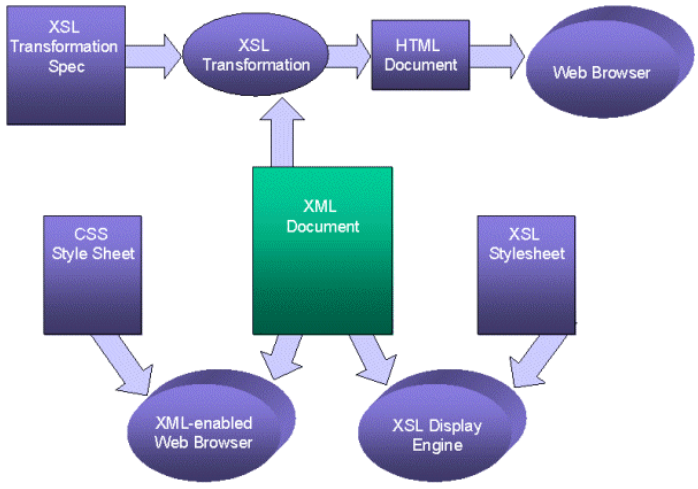

XSL
What's with stylesheets in the first place?

- XML is not a fixed tag set (like HTML)
- XML by itself has no (application) semantics
- A generic XML processor has no idea what is "meant" by the XML
- XML markup does not (usually) include formatting information
- The information in an XML document may not be in the form in which it is desired to present it

Advantages to separating content from style

- reuse of fragments of data
- multiple output formats
- styles tailored to the reader's preference (e.g., accessibility
- standardized styles: corporate stylesheets can be applied to the content at any time
- freedom from style issues for content authors

Options for displaying XML



What Does a Stylesheet Do?

A stylesheet specifies the presentation of XML information using two basic categories of techniques:

- An optional transformation of the input document into another structure
- A description of how to present the transformed information (i.e., a specification of what properties to associate to each of the various parts of the transformed information)

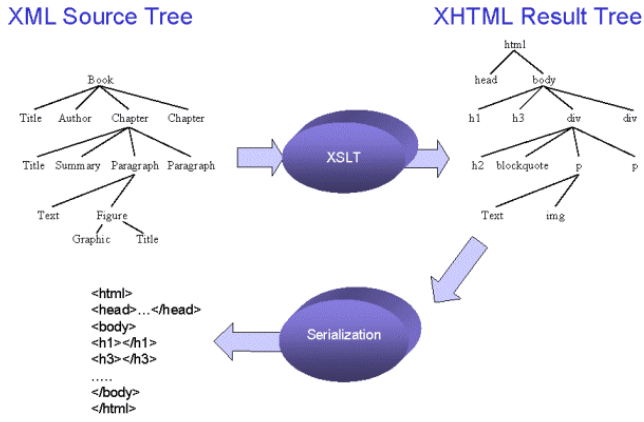
The components of the XSL language

The full XSL language logically consists of three component languages which are described in three W3C Recommendations:

- *XPath*: XML Path Language--a language for referencing specific parts of an XML document
- *XSLT*: XSL Transformations--a language for describing how to transform one XML document (represented as a tree) into another
- *XSL*: Extensible Stylesheet Language--XSLT plus a description of a set of Formatting Objects and Formatting Properties

XML to result tree

An XSLT transforms the input document's tree into a structure called a *result tree* consisting of result objects



An XSL stylesheet

- An XSL stylesheet basically consists of a set of templates
- Each template "matches" some set of elements in the source tree and then describes the contribution that the matched element makes to the result tree

HTML vs. XSL Formatting Objects

- Transformation is independent of the target result type
- Most people are more familiar with HTML so many of the examples in this tutorial use HTML
- The XSL implementation in IE5 is incomplete. The examples in this tutorial will not work in IE5
- The techniques apply equally well to XSL Formatting Objects or other tag sets
- XSLT is a tree-to-tree transformation process
- Serialization may vary depending on the selected output method

The Structure of a Stylesheet

- XSLT Stylesheets are XML documents; namespaces are used to identify semantically significant elements.
- Most stylesheets are stand-alone documents rooted at <xsl:stylesheet> or <xsl:transform>. It is possible to have "single template" stylesheet/documents.
- <xsl:stylesheet> and <xsl:transform> are completely synonymous.

Stylesheet Examples...

- **A Stylesheet**

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

- **A Transformation Sheet**

```
<eg:transform xmlns:eg="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
    ...
</eg:transform>
```


..Stylesheet Examples

- **With this stylesheet**

```
<?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>
```

- **the following XML document**

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

- **Would be transformed into**

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

Sorting

The <xsl:sort> element sorts a set of nodes according to the criteria specified:

XSL :Example1...

- **XML file**

```
<PRODOTTO>
    <IMMAGINE MINIATURA="bell_120.jpg" INTERA="bell_full.jpg" TESTO="Barbarella"/>
    <DESCRIZIONE>Bambolina
            <EM>Barbarella</EM> snodata e con accessori (spazzola e pettine).
    </DESCRIZIONE>
    <PREZZO VALUTA="EUR">8,25</PREZZO>
</PRODOTTO>
```

XSL :... Example 1

- **XSL file**

```
<xsl:template match="PRODOTTO">
    <TR>
      <TD xsl:use"img-cell-attr">
        <A HREF="{IMMAGINE/attribute(INTERA)}" TARGET="_top">
          <IMG SRC="{IMMAGINE/attribute(MINIATURA)}" BORDER=0
              ALT="{IMMAGINE/attribute(TESTO)}">
        </A>
      </TD>
      <TD xsl:use"dsc-cell-attr">
        <xsl:value-of expr="DESCRIZIONE"/>
      </TD>
      <TD xsl:use"prz-cell-attr">
        <xsl:value-of expr="PREZZO/attribute(VALUTA)"/>
        <xsl:text> </xsl:text>
        <xsl:value-of expr="PREZZO"/>
      </TD>
    </TR>
</xsl:template>
```


XSL :Example 2...

- **XML file**

```
<rivista>
    <articolo>
            <titolo> XSLT e XPATH</titolo>
            <autore>Moris Bellavista</autore>
            <numero>25</numero>
            <descrizione>Nel precedente articolo abbiamo introdotto l'XML ….</descrizione>
            <data>10/07/2001</data>
            <bibliografia>
                    <riferimento numero="1">Paul Grosso e Norman Walsh</riferimento>
                    <riferimento numero="2">Michael Kay….Reference</riferimento>
                    <riferimento numero="3">G. Ken Holman…..</riferimento>
            </bibliografia>
            <note_autore>Moris Bellavista e' laureato in Scienze dell'Informazione…..</note_autore>
            <e-mail>bellavista@infomedia.it</e-mail>
    </articolo>
</rivista>
```

XSL :.. Example2..

- **XSL file**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">
        <html>
            <body bgcolor="#ffffff">
                <xsl:apply-templates select="rivista/articolo"/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="articolo">
        <h1> Titolo: <xsl:value-of select="titolo"/> </h1>
        <h2> di <xsl:value-of select="autore"/></h2>
        ……..
        <h4>Bibliografia:</h4>
        <table> <xsl:apply-templates select="bibliografia"/></table>
        <h4>
            <a>
              <xsl:attribute name="href">mailto:<xsl:value-of select="e-mail"/></xsl:attribute>
              <xsl:value-of select="e-mail"/>
            </a>
        </h4>
    </xsl:template>
```


XSL :.. Example2

```
    <xsl:template match="bibliografia">
        <xsl:for-each select="riferimento">
            <tr>
                <td>
                    [<xsl:value-of select="@numero"/>]
                </td>
                <td>
                    <xsl:value-of select="."/>
                </td>
            </tr>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Further reading

http://www.w3.org/XML

Programmare il World Wide Web
Robert W. Sebesta – *Capitolo 10*

http://www.xmlspy.com