XML

5 October 2016

Bob Glushko

Plan for the Lecture

Syntax and Structure Matters!

Minimal XML Syntax Introduction

Document Types

Implementing Document Types in XML

XML Transformation

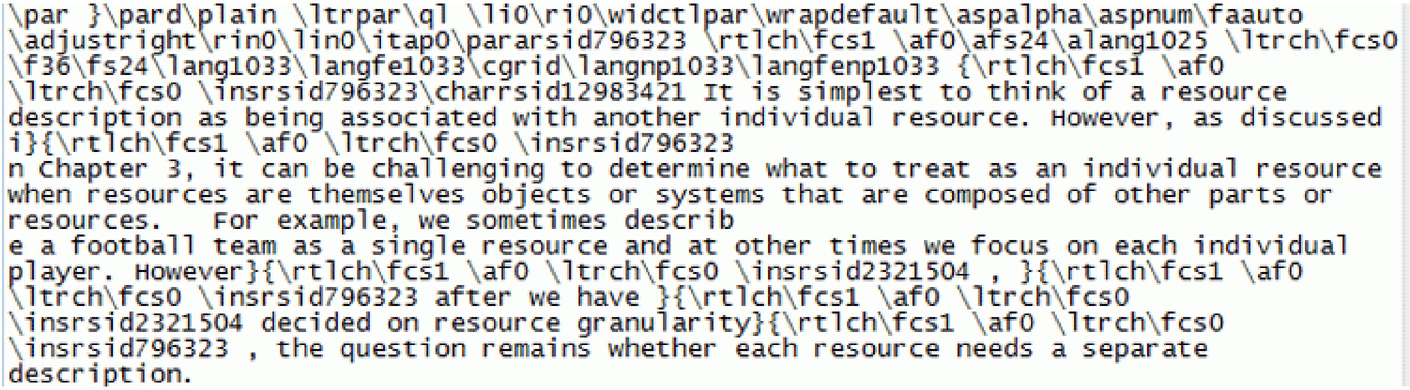
Separating Content from Container

Not all information content can be completely separated from its container (sometimes the medium is the message)

But it is important to think of the information content abstractly if you can because that's the key to representing the same information in multiple formats, media, or technologies

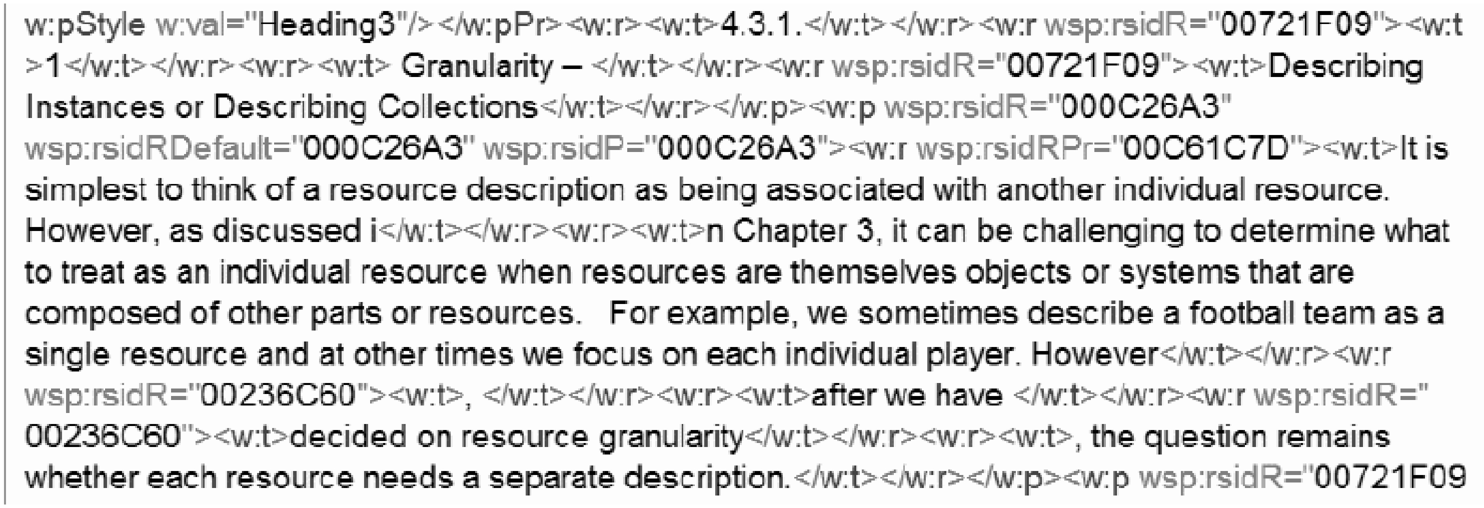
Some information formats or representations are inherently more processable or reusable than others; this reflects how completely they separate content from presentation

Microsoft Word "Rich Text Format"



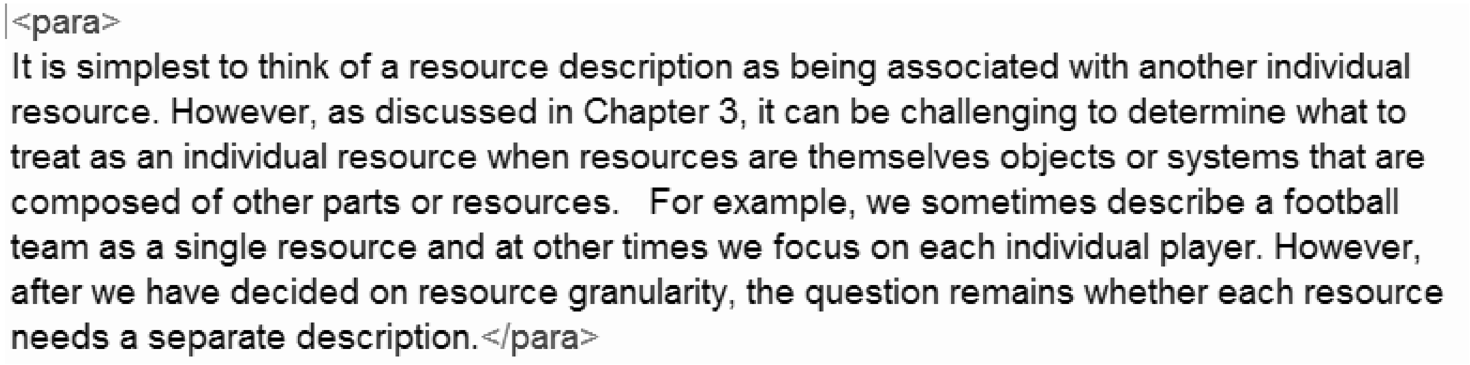
```
\par }\pard\plain \ltrpar\ql \li0\ri0\widctlpar\wrapdefault\aspalpha\aspnum\faauto
\adjustright\rin0\lin0\itap0\pararsid796323 \rtlch\fcs1 \af0\afs24\alang1025 \ltrch\fcs0
\f36\fs24\lang1033\langfe1033\cgrid\langnp1033\langfenp1033 {\rtlch\fcs1 \af0
\ltrch\fcs0 \insrsid796323\charrsid12983421 It is simplest to think of a resource
description as being associated with another individual resource. However, as discussed
i}{\rtlch\fcs1 \af0 \ltrch\fcs0 \insrsid796323
n Chapter 3, it can be challenging to determine what to treat as an individual resource
when resources are themselves objects or systems that are composed of other parts or
resources.   For example, we sometimes describ
e a football team as a single resource and at other times we focus on each individual
player. However}{\rtlch\fcs1 \af0 \ltrch\fcs0 \insrsid2321504 , }{\rtlch\fcs1 \af0
\ltrch\fcs0 \insrsid796323 after we have }{\rtlch\fcs1 \af0 \ltrch\fcs0
\insrsid2321504 decided on resource granularity}{\rtlch\fcs1 \af0 \ltrch\fcs0
\insrsid796323 , the question remains whether each resource needs a separate
description.
```

Microsoft Word XML



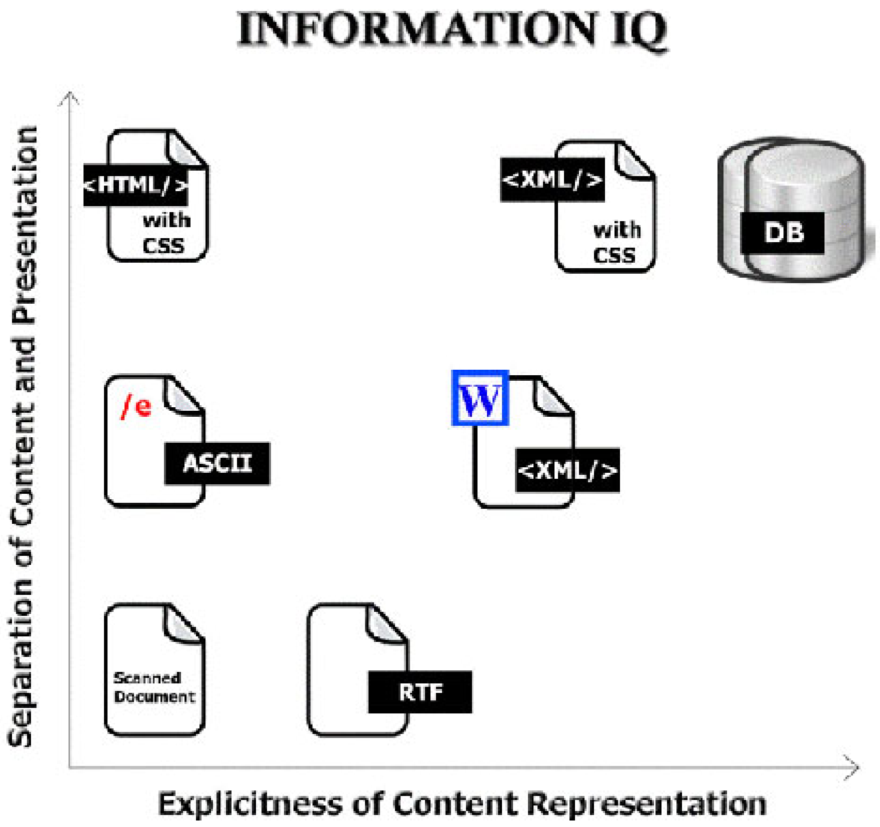
```
w:pStyle w:val="Heading3"/></w:pPr><w:r><w:t>4.3.1.</w:t></w:r><w:r wsp:rsidR="00721F09"><w:t
>1</w:t></w:r><w:r><w:t> Granularity – </w:t></w:r><w:r wsp:rsidR="00721F09"><w:t>Describing
Instances or Describing Collections</w:t></w:r></w:p><w:p wsp:rsidR="000C26A3"
wsp:rsidRDefault="000C26A3" wsp:rsidP="000C26A3"><w:r wsp:rsidRPr="00C61C7D"><w:t>It is
simplest to think of a resource description as being associated with another individual resource.
However, as discussed i</w:t></w:r><w:r><w:t>n Chapter 3, it can be challenging to determine what
to treat as an individual resource when resources are themselves objects or systems that are
composed of other parts or resources.   For example, we sometimes describe a football team as a
single resource and at other times we focus on each individual player. However</w:t></w:r><w:r
wsp:rsidR="00236C60"><w:t>, </w:t></w:r><w:r><w:t>after we have </w:t></w:r><w:r wsp:rsidR="
00236C60"><w:t>decided on resource granularity</w:t></w:r><w:r><w:t>, the question remains
whether each resource needs a separate description.</w:t></w:r></w:p><w:p wsp:rsidR="00721F09
```

XML -- Assuming a Style Sheet



```
<para>
It is simplest to think of a resource description as being associated with another individual
resource. However, as discussed in Chapter 3, it can be challenging to determine what to
treat as an individual resource when resources are themselves objects or systems that are
composed of other parts or resources.   For example, we sometimes describe a football
team as a single resource and at other times we focus on each individual player. However,
after we have decided on resource granularity, the question remains whether each resource
needs a separate description.</para>
```

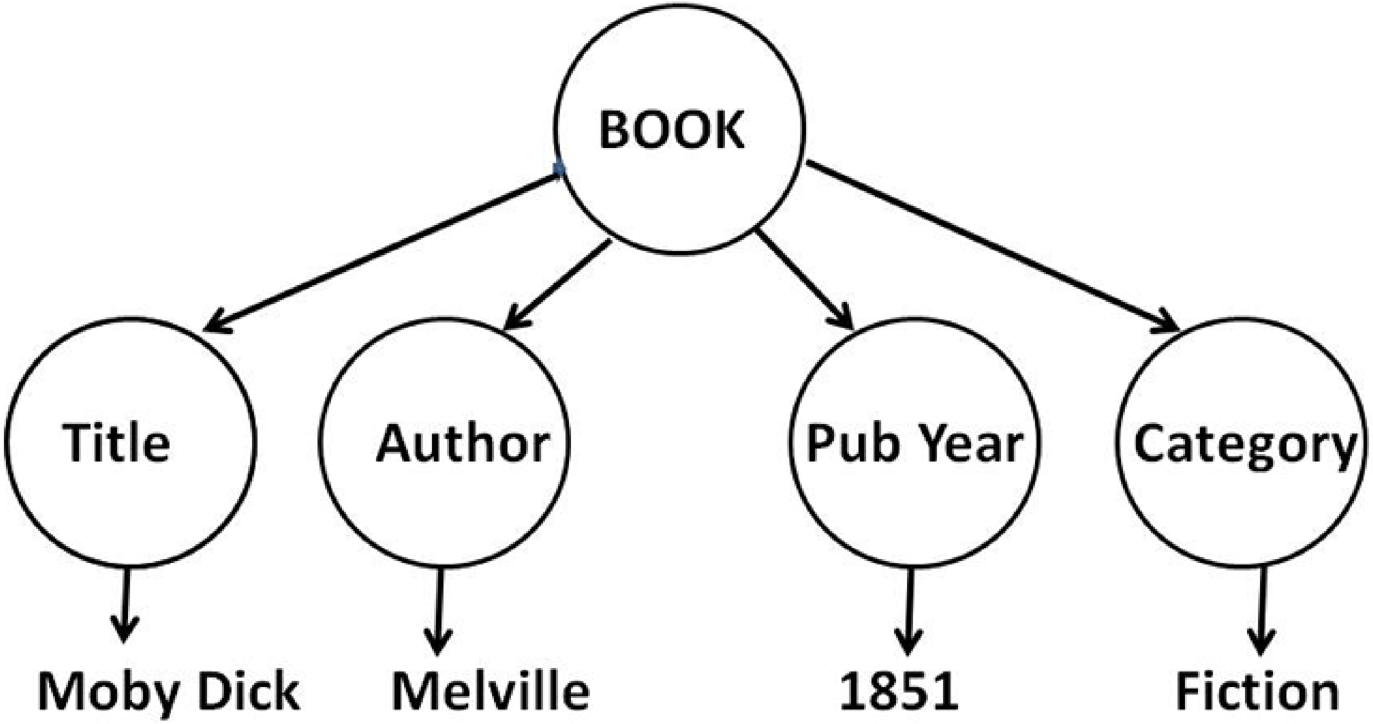
"Information IQ"



Unstructured Information as Sentence Blob

Moby Dick is a fiction book written by Melville in 1851

Structure as Tree



HTML Implementation of a Document Modeled as Tree

```
<p>
<ul>
  <li>Moby Dick</li>
  <li>Melville</li>
  <li>1851</li>
  <li>Fiction</li>
</ul>
</p>
```

An XML Implementation of same Document Model

```
<Book>
  <Title>Moby Dick</Title>
  <Author>Melville</Author>
  <PublicationYear>1851</PublicationYear>
  <Category>Fiction</Category>
</Book>
```

Comparing HTML and XML

Superficial Similarity in Syntax: Both use

- Begin and End Tags
- Containers

But XML

- Isn't a specific "tag language" like HTML is
- Is a metalanguage that can define new "tag languages"
- Changes the task from "how do I make this web page look" to "what is this web page about?"

A Purchase Order in HTML

```
<HTML>
<BODY>
<H1>Purchase Order (#1234)</H1>
<HR>
<H2>Buyer Information</H2>
<P>Smith and Company (Buyer # AB24567)</BR>
123 High St., Suite 100</BR>
Anytown, California 12345
</P>
<HR>
<H2>Items</H2>
<OL>
<LI>20 Widgets
</OL>
</BODY>
</HTML>
```

A Purchase Order in XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder OrderNo="1234">
        <Buyer BuyerNo="AB24567">
                <Name>Smith and Company</Name>
                <Address1>123 High Street</Address1>
                <Address2>Suite 100</Address2>
                <City>Anytown</City>
                <State>California</State>
                <ZipCode>12345</ZipCode>
        </Buyer>
        <Items>
                <Item>
                        <Quantity>10</Quantity>
                        <ItemName>Widget</ItemName>
                </Item>
                <Item>
                        <Quantity>20</Quantity>
                        <ItemName>Bazooka</ItemName>
                </Item>
        </Items>
</PurchaseOrder>
```

HTML is a Model of a Document Type

HTML's idea of using tags to mark up pieces of text according to how they should appear on the page for people to read them is a simple model of a document

This model (until HTML5) emphasizes structures like headings and lists, and presentation or formatting

This model specifies a FIXED set of element types ("tags") and attributes that a document can contain

Because the set of tags is fixed, browsers can implement all of them with default presentation and behavior

XML is a Metamodel for Document Types

XML is *a syntax for encoding domain-specific models and instances* in ways that can be handled by applications

So XML is a METALANGUAGE that can be used to define new languages; XHTML is an example of one, as is the "Book" language here

What kind of information (types of elements) is needed to encode a model of:

- A Shakespeare play?
- A chemical molecule?

Because an XML vocabulary can contain any tag, browsers can't be hard-wired to render them in any particular way

Shakespeare in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<PLAY>
<TITLE>The Tragedy of Hamlet, Prince of Denmark</TITLE>
<PERSONAE>
<TITLE>Dramatis Personae</TITLE>
<PERSONA>CLAUDIUS, king of Denmark. </PERSONA>
<PERSONA>HAMLET, son to the late, and nephew to the present king.</PERSONA>
<PERSONA>POLONIUS, lord chamberlain. </PERSONA>
...
</PERSONAE>
<ACT><TITLE>ACT III</TITLE>
<SCENE><TITLE>SCENE I.  A room in the castle.</TITLE>
<STAGEDIR>Exeunt KING CLAUDIUS and POLONIUS</STAGEDIR>
<STAGEDIR>Enter HAMLET</STAGEDIR>
<SPEECH>
  <SPEAKER>HAMLET</SPEAKER>
  <LINE>To be, or not to be: that is the question:</LINE>
  <LINE>Whether 'tis nobler in the mind to suffer</LINE>
  <LINE>The slings and arrows of outrageous fortune,</LINE>
  <LINE>Or to take arms against a sea of troubles,</LINE>
    ...
    </SPEECH>
  </SCENE>
</ACT>
</PLAY>
```

Chemistry in XML

```
<cml xmlns:stm="http://www.xml-cml.org/schema/cml2/core"
        xmlns="http://www.xml-cml.org/schema/cml2/core">
  <substanceList id="sl1">
    <amount units="g">4.0</amount>
    <substance id="s1"></substance>
  </substanceList>
    <crystal>
     <stm:scalar title="a">12.34</stm:scalar>
     ...
      <stm:scalar title="gamma" units="degrees">100</stm:scalar>
      <symmetry id="ss1" spaceGroup="P21/c"/>
    </crystal>
    <atomArray>
      <atom id="a1" elementType="C">
        <atomParity atomRefs4="a1 a2 a3 a4">-1.0</atomParity>
        <electron id="e1"></electron>
      </atom>
      <atom id="a2" elementType="C"/>
      ...
    </atomArray>
...
  </molecule>
  </cml>
```

Smart Markup Creates Explicit Text Objects

How much you can do with information depends on the extent and explicitness of its internal structure or "markup"

Markup transforms a flat stream or blob of text into a set of objects or elements that can be manipulated by other applications

HTML is rich or smart compared with some text formats like EDI, RTF, or ASCII

And all of these syntaxes are poor or dumb compared with a content-oriented data model, such as those possible with XML, SGML, or a database

Elements and Containers

```
<BookTitle>Moby Dick</BookTitle>
<BookAuthor>Melville</BookAuthor>
<BookPublicationYear>1851</BookPublicationYear>
<BookCategory>Fiction</BookCategory>
```

```
<Book>
  <Title>Moby Dick</Title>
  <Author>Melville</Author>
  <PublicationYear>1851</PublicationYear>
  <Category>Fiction</Category>
</Book>
```

Elements are the building blocks in XML documents

They define the hierarchy or logical "containers" by enclosing content with both a begin and end tag; the hierarchy provides context for understanding the child elements

Attributes

Elements may also one or more attributes (a name - value pair) associated only with their start tag and the values must always be quoted with matching ' or "

- <PurchaseOrder number="12" >*purchase order content* </PurchaseOrder>
- The order of attributes is not significant

Elements with attributes but no content are said to be "empty" and have a different tag syntax

- <Portrait image="bob.gif"/>

Elements {and,or,vs} Attributes

Whether to use elements or attributes to contain information is often debated

Elements and attributes differ in what they can contain and this often guides which you should use

Elements can contain other elements while attributes can contain only strings or lists of strings

So elements must be used for any complex components but can be used for simple or primitive ones as well; attributes can only be used for "atomic" items of data

Elements carry the content that would generally appear in any presentation or rendering of the XML instance; attributes carry "strong" metadata or information that is useful in interpreting or presenting the element content

Elements {and,or,vs} Attributes [2]

Attributes are the only way to specify default values and can be constrained to a predefined set of enumerated values

Attributes are also the most sensible way to encode Boolean values

Attributes are inconvenient for long text, large values, or binary entities

If information is primarily encoded as attributes the XML instance can be significantly smaller

"Best practice" is contentious but many people use almost all elements and very few attributes, leaving the latter for just the "purest" metadata

Elements {and,or,vs} Attributes [3]

```
<Book>
  <Title>Moby Dick</Title>
  <Author>Melville</Author>
  <PublicationYear>1851</PublicationYear>
  <Category>Fiction</Category>
</Book>
```

```
<Book title="Moby Dick" author="Melville"  publicationYear="1851"  category="Fiction"/>
```

```
<Book title="Moby Dick" author="Melville" category="Fiction" publicationYear="1851"/>
```

```
<Book title="Moby Dick" author="Melville"  publicationYear="1851"  fiction="True"/>
```

Elements {and,or,vs} Attributes [4]

```
<Book>
  <Title>Moby Dick</Title>
  <Author>Melville</Author>
  <PublicationYear>1851</PublicationYear>
   <Categories>
      <Category>Fiction<Category>
      <Category>Whales<Category>
  </Categories>
</Book>
```

This is legal but not as useful

```
<Book title="Moby Dick"
       author="Melville"
       publicationYear="1851"
       category="Fiction  Whales"  />
```

Elements {and,or,vs} Attributes [5]

```
<Book>
  <Title>Moby Dick</Title>
  <Author>Melville</Author>
  <PublicationYear>1851</PublicationYear>
   <Categories>
      <Category>Fiction<Category>
      <Category>Whales<Category>
  </Categories>
</Book>
```

But this is illegal

```
<Book title="Moby Dick"
       author="Melville"
       publicationYear="1851"
       category="Fiction"
       category="Whales" />
```

Document Types

Any definition of "document" allows for a notion of different types or classes or categories of documents

This idea can be very intuitive and very informal, or we can be more precise and define a MODEL OF A DOCUMENT TYPE as the rules or constraints that distinguish one type from another

This expression of the model is CONCEPTUAL and is independent of the syntax and technology in which document instances are ultimately implemented

But most of the time the model is ultimately implemented in some specific syntax like XML

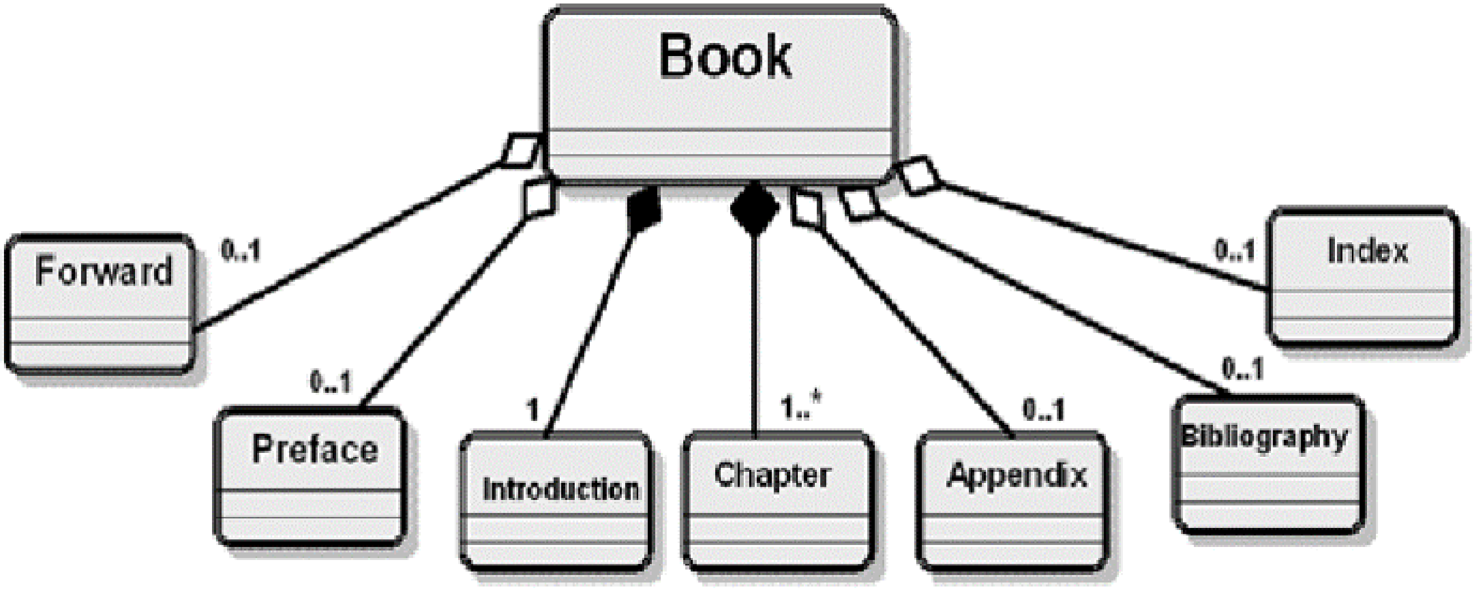
Models of Document Types

A model of a document type captures the distinctions between documents that make a difference

Similar types of content occur in many document models and there is often overlap in information and structural patterns

Models of document types can be very specific ("purchase order for industrial chemicals when buyer and seller are in different countries") or very abstract ("fill-in-the-blank legal form for contract")

Model of a "Book" Document Type



(Finally) How XML Implements Models of Document Types

XML gives the idea of document type a more physical, formal foundation

XML has syntactic mechanisms that capture the conceptual distinctions between document types in terms of:

- ELEMENTS (the "tags") and ATTRIBUTES used to encode their content
- Rules that govern how elements and attributes are organized
- Possible values for elements and attributes

These are the VOCABULARY and the GRAMMAR of the language defined by the document type

Using XML to Encode Document Type Models

Encoding a conceptual information model in XML means choosing elements or attributes as the containers for information, adding information about data types, applying naming rules, creating structures to organize repeated content components, and so on

If you've done a careful document analysis and design, the encoding stage is relatively simple and straight-forward and can even be automated in some cases

XML Schemas

The formal description of a document model in XML is called its *schema*

XML schemas (lower case "s" for now) attempt to encode the conceptual model in terms of the syntactic constructs of elements and attributes

- What elements are allowed (the *vocabulary*)
- Where the are allowed – sequence, choice, occurrence and co-occurrence (the *grammar*)
- What values they can take (*datatypes*) – string, integer, decimal, etc.

Why We Need Schemas

If you can represent these rules that define a document type in a form that is "computable" or "processable" then:

- It can guide the creation of valid document instances in editors like XML Spy or Oxygen, or when information is exported from a database or other application
- It can be a model for application programming in the development of Web forms or other GUIs or can be a template for objects in other programming environments
- It can communicate the model to others who need to create or receive document instances

XML Schema Languages

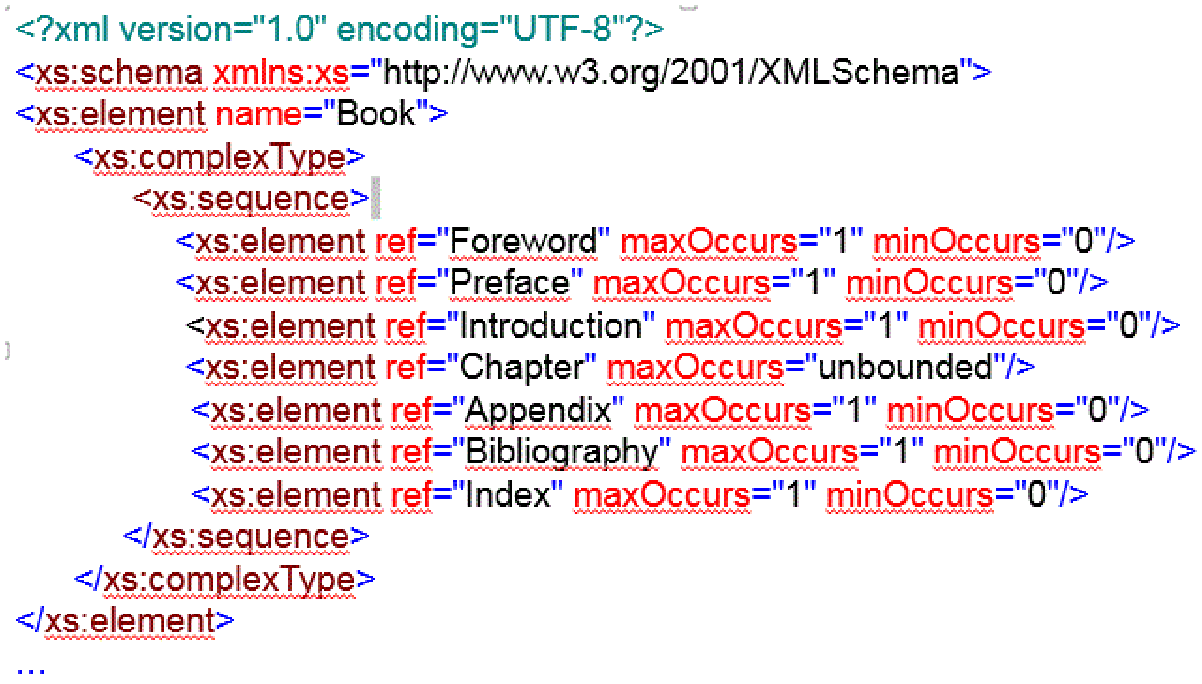
XML has several schema languages that differ in how completely that can encode a document type's conceptual model

The most common of these are Document Type Definitions (DTDs) and XML Schemas (XSD)

No schema language is perfect; there is always some compromise between:

- Expressiveness – the range of models that can be described
- Functionality – the set of features used to define a model
- Usability – the ease with which a model can be defined
- Reusability – how readily a model or parts of models can be included in another one
- ...and a range of other "ilities"

XSD Schema for "Book" Document Type



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Book">
    <xs:complexType>
        <xs:sequence>
           <xs:element ref="Foreword" maxOccurs="1" minOccurs="0"/>
           <xs:element ref="Preface" maxOccurs="1" minOccurs="0"/>
            <xs:element ref="Introduction" maxOccurs="1" minOccurs="0"/>
            <xs:element ref="Chapter" maxOccurs="unbounded"/>
             <xs:element ref="Appendix" maxOccurs="1" minOccurs="0"/>
             <xs:element ref="Bibliography" maxOccurs="1" minOccurs="0"/>
             <xs:element ref="Index" maxOccurs="1" minOccurs="0"/>
       </xs:sequence>
    </xs:complexType>
</xs:element>
…
```

Why Transform?

You have information that is too "smart" for the web or end users to handle (for example, a database that will be queried, a complex information model for which simpler views are needed for some users)

You need to RE-PURPOSE information – extracting and / or formatting the same piece of information is many different ways, producing a different document type targeted for a different user or purpose

You have to support a variety of output devices that have different capabilities– often called RE-PACKAGING or TAILORING

You need to conform to a structural or formatting standard that is different from your company or organization's information model

Your "web service" needs to convert an "inbound" non-XML document to XML, or convert XML to a non-XML format for the "outbound" document

Technical Example of Re-purposing – Product Database / Catalog

The XML instance: product-db.xml

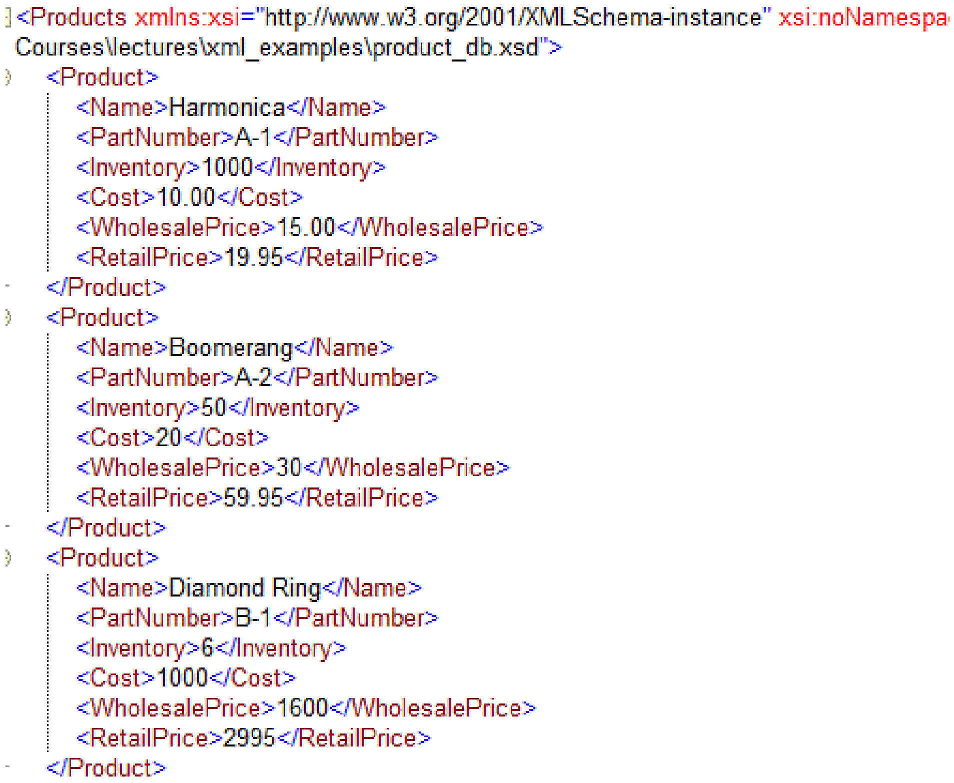
The target HTML instance viewed as an internal database: product-db.htm

The target HTML instance viewed as a catalog for external customers: product-catalog.htm

The XSLT transformation for the database: product_db.xsl

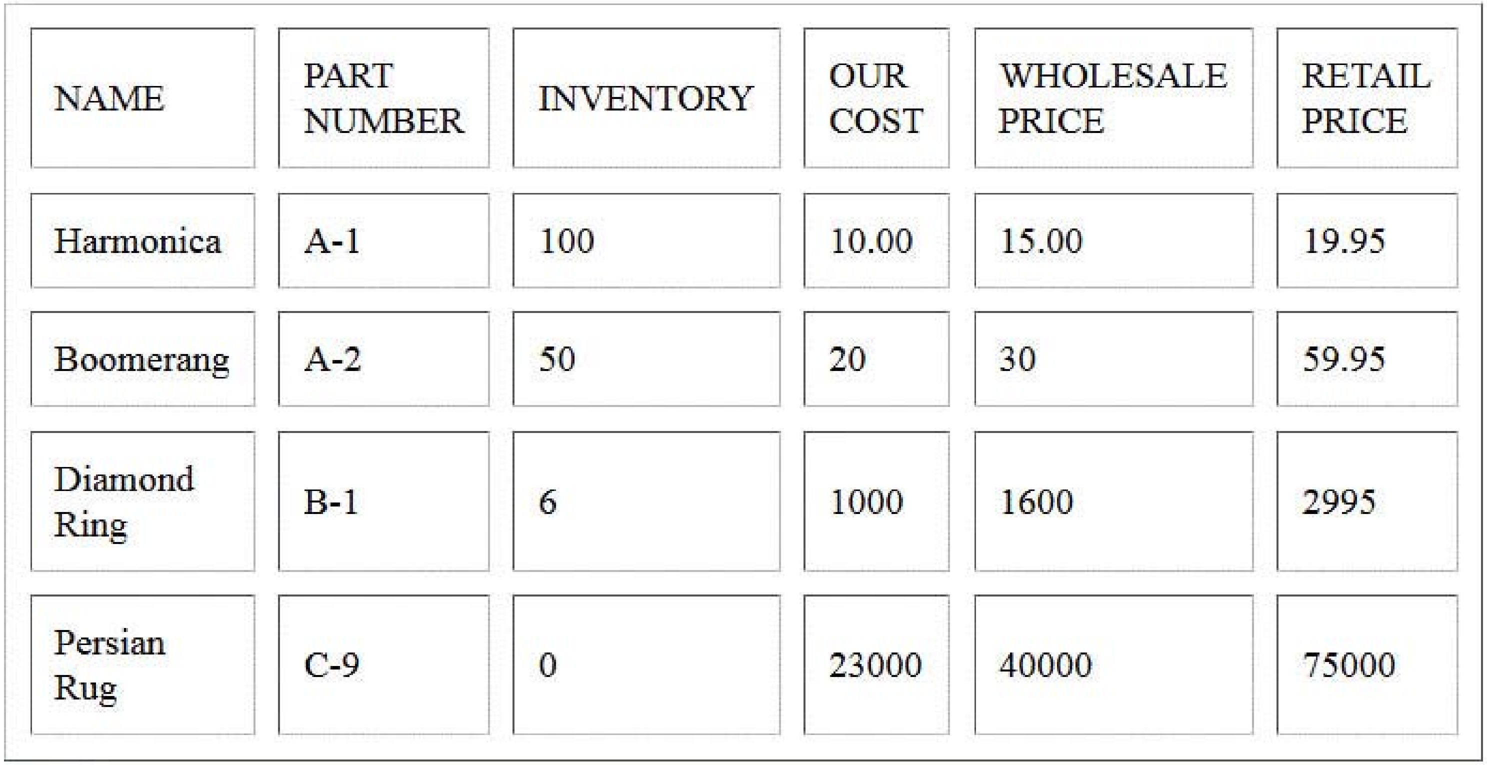
The XSLT transformation for the catalog: product_catalog.xsl

XML Instance



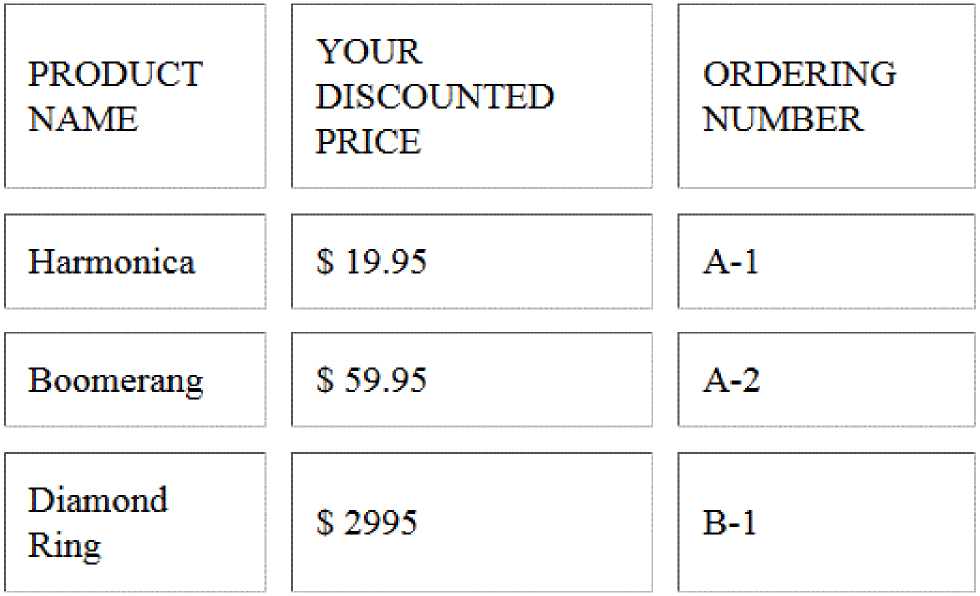
```
<Products xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespa
Courses\lectures\xml_examples\product_db.xsd">
    <Product>
        <Name>Harmonica</Name>
        <PartNumber>A-1</PartNumber>
        <Inventory>1000</Inventory>
        <Cost>10.00</Cost>
        <WholesalePrice>15.00</WholesalePrice>
        <RetailPrice>19.95</RetailPrice>
    </Product>
    <Product>
        <Name>Boomerang</Name>
        <PartNumber>A-2</PartNumber>
        <Inventory>50</Inventory>
        <Cost>20</Cost>
        <WholesalePrice>30</WholesalePrice>
        <RetailPrice>59.95</RetailPrice>
    </Product>
    <Product>
        <Name>Diamond Ring</Name>
        <PartNumber>B-1</PartNumber>
        <Inventory>6</Inventory>
        <Cost>1000</Cost>
        <WholesalePrice>1600</WholesalePrice>
        <RetailPrice>2995</RetailPrice>
    </Product>
```

Internal Database View



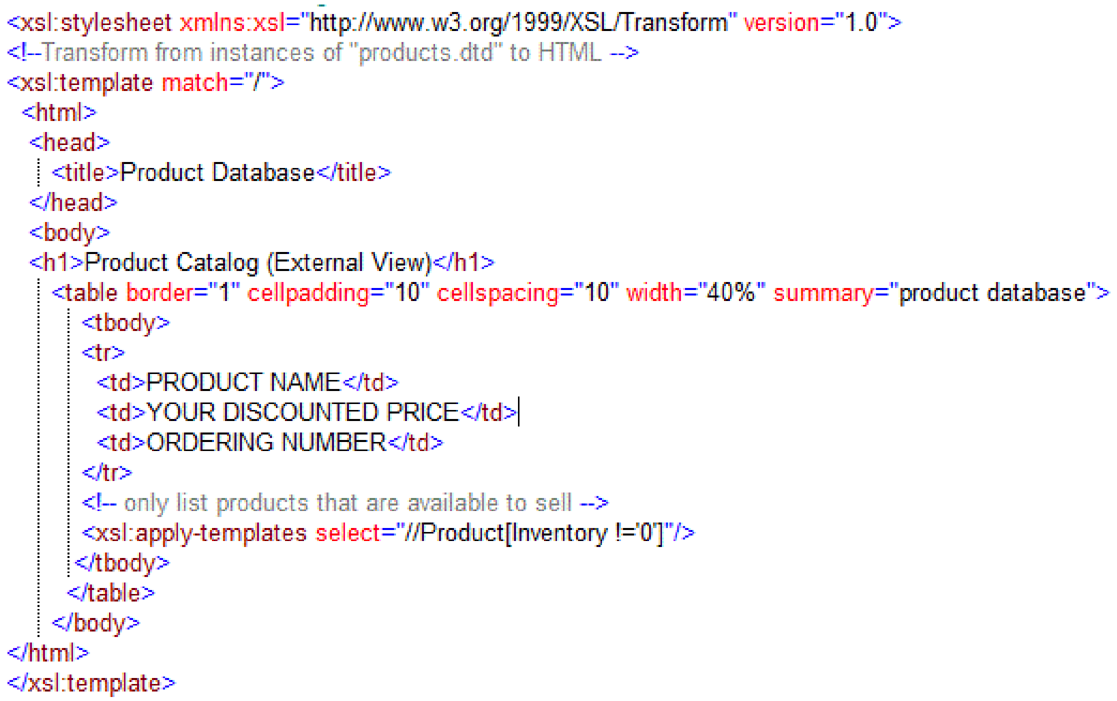
NAME	PART NUMBER	INVENTORY	OUR COST	WHOLESALE PRICE	RETAIL PRICE
Harmonica	A-1	100	10.00	15.00	19.95
Boomerang	A-2	50	20	30	59.95
Diamond Ring	B-1	6	1000	1600	2995
Persian Rug	C-9	0	23000	40000	75000

External Catalog View



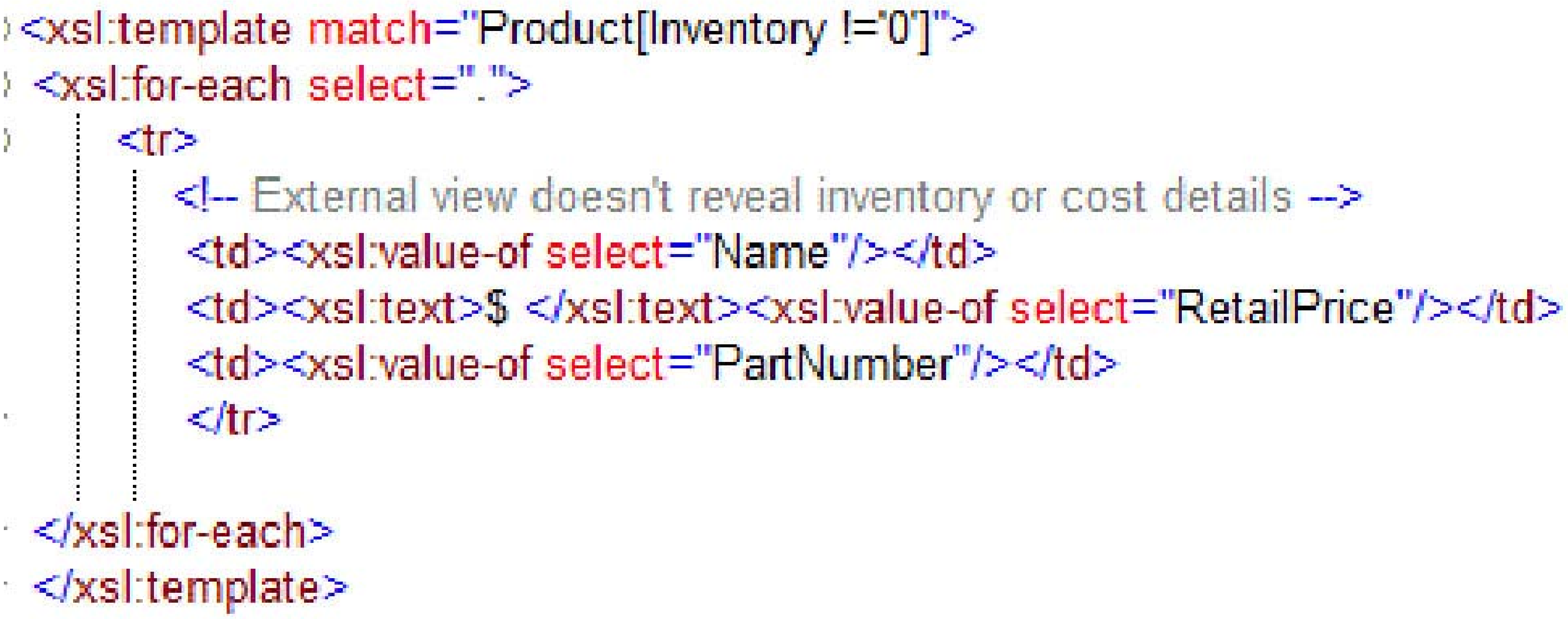
PRODUCT NAME	YOUR DISCOUNTED PRICE	ORDERING NUMBER
Harmonica	$ 19.95	A-1
Boomerang	$ 59.95	A-2
Diamond Ring	$ 2995	B-1

The XSL Transform - Creating the Product Table



```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<!--Transform from instances of "products.dtd" to HTML -->
<xsl:template match="/">
 <html>
  <head>
   <title>Product Database</title>
  </head>
  <body>
  <h1>Product Catalog (External View)</h1>
   <table border="1" cellpadding="10" cellspacing="10" width="40%" summary="product database">
     <tbody>
     <tr>
      <td>PRODUCT NAME</td>
      <td>YOUR DISCOUNTED PRICE</td>
      <td>ORDERING NUMBER</td>
     </tr>
     <!-- only list products that are available to sell -->
     <xsl:apply-templates select="//Product[Inventory !='0']"/>
    </tbody>
    </table>
   </body>
</html>
</xsl:template>
```

The XSL Transform - Only the Available Products



```
<xsl:template match="Product[Inventory !='0']">
 <xsl:for-each select=".">
      <tr>
          <!-- External view doesn't reveal inventory or cost details -->
           <td><xsl:value-of select="Name"/></td>
           <td><xsl:text>$ </xsl:text><xsl:value-of select="RetailPrice"/></td>
           <td><xsl:value-of select="PartNumber"/></td>
           </tr>

 </xsl:for-each>
 </xsl:template>
```

The Architecture of XML Transformation

A particular transformation may apply to more than one document – it might be used to enforce standards for all instances of a document type

```
<?xml-transform type="text/xsl" href="standard_style_for_this_doctype.xsl"?>
```

A given document instance may have different transforms applied to it in different contexts (like for different audiences, output devices, etc)

```
<?xml-transform type="text/xsl" href="transform_for_smartphone.xsl"?>
<?xml-transform type="text/xsl" href="transform_for_verbose_mode.xsl"?>
```

A transformation may turn one XML file into one or more output files (like the XML source file for each lecture being transformed into a set of linked HTML slides)