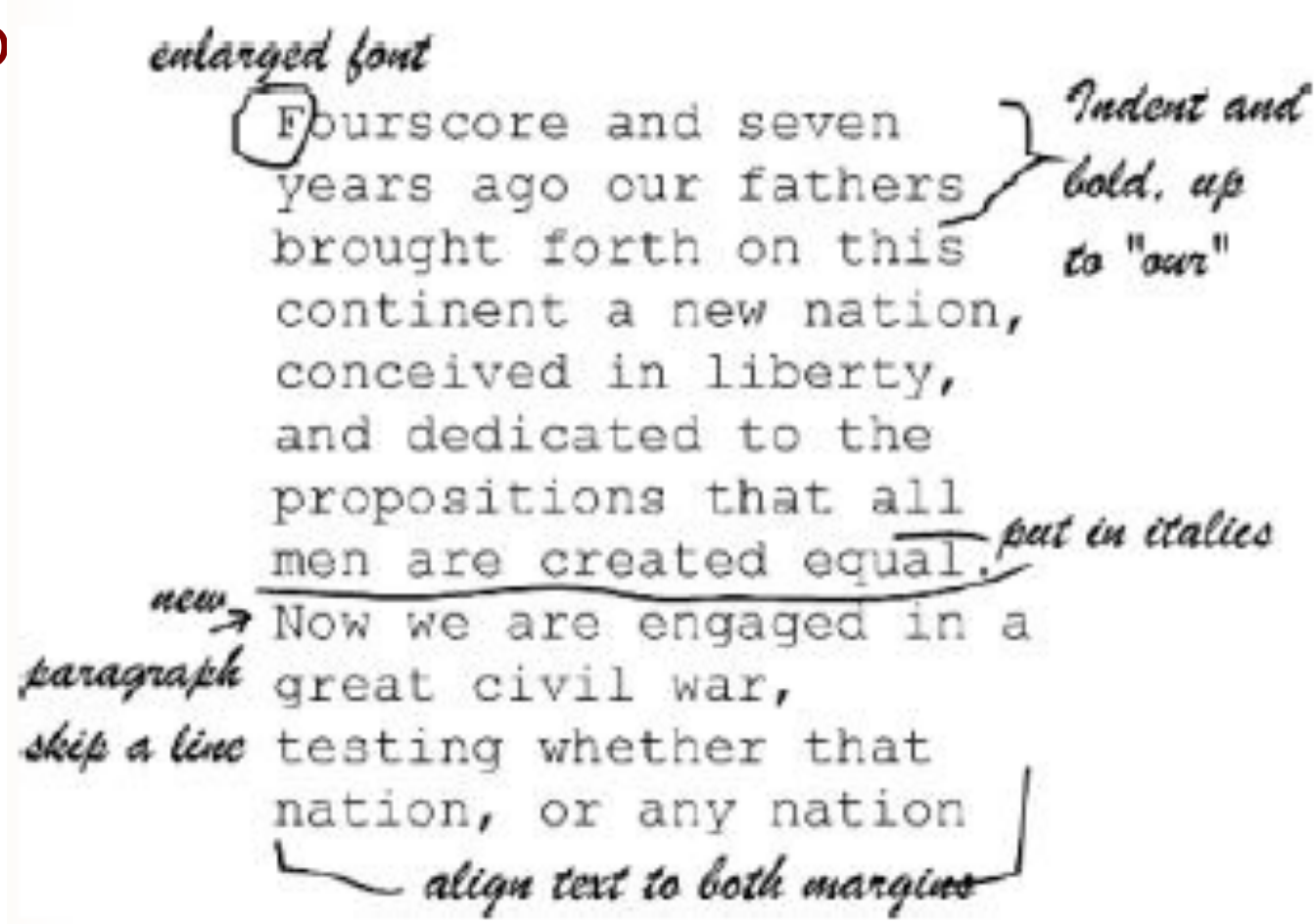
XML, DTD, XML Schema

Some slides are from Roger Costello

What is XML (eXtremely Marketed Language)

- Markup



From XML Handbook

XML (eXtensible Markup Language)

- Markup in XML
  - A sequence of characters inserted into a text file,
    - to indicate how the file should be displayed, or
    - to describe the logical structure.
  - Markup is everything in a document that is not content.
  - Initially used in typesetting a document
  - Markup indicators are called *tag*s. e.g.

    ```
    <font color="blue">
    ```

  - A pair of tags and the things enclosed in tags is called *element*. e.g.

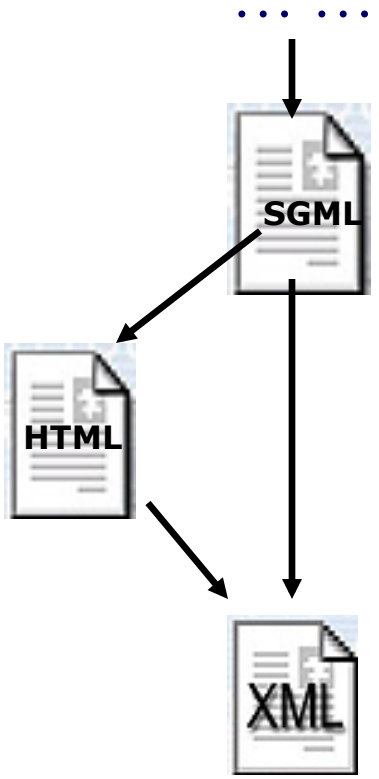
    ```
    <font color="blue"> formatted as blue </font>
    ```

What is XML (eXtensible Markup Language) (cont.)

- Extensible
  - In general: Something that is designed that users or later designers can extend its capability.
  - In XML: Allow you to define your own tags to describe data
    - You can represent any information (define new tags)
    - You can represent in the way you want (define new structure)
  - XML is a meta-language
    - A language to define other languages
    - Use DTD/XML Schema to define the syntax of a language

Markup (and extensible) languages are not new

- SGML (Standard Generalized Markup Language)
  - Markup, extensible
  - 1980: first publication, 1986: ISO standard
- HTML(HyperText Markup Language)
  - Markup, hypertext, Subset of SGML
  - Started 1990, CERN (Centre Européen de Recherche Nucléaire, or European High-Energy Particle Physics lab)
  - Invented by Tim Berners-Lee
- XML (eXtensible Markup Language)
  - Subset of SGML
  - Started 1996, adopted by W3C 1998
  - Eliminate the complexity of SGML
  - Separate the data from the formatting information in HTML



HTML

- table: <table>
- Table head: <TH>
- Table row: <TR>
- Table data: <TD>

open tag, element name

attribute

Attribute value

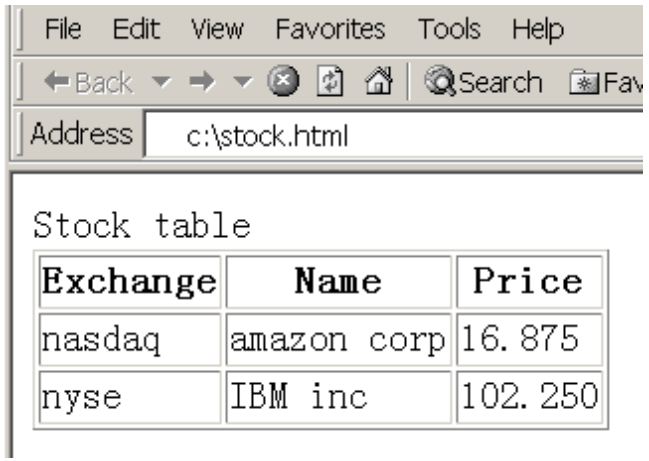
```
<html><body>  Stock table
<TABLE border="1">
<TR><TH> Exchange </TH> <TH>  Name        </TH> <TH> Price  </TH> </TR>
<TR><TD> nasdaq   </TD> <TD> amazon corp </TD> <TD> 16.875 </TD> </TR>
<TR><TD> nyse     </TD> <TD> IBM inc     </TD> <TD> 102.250</TD> </TR>
</TABLE> </body></html>
```

stock.html

closing tag

data

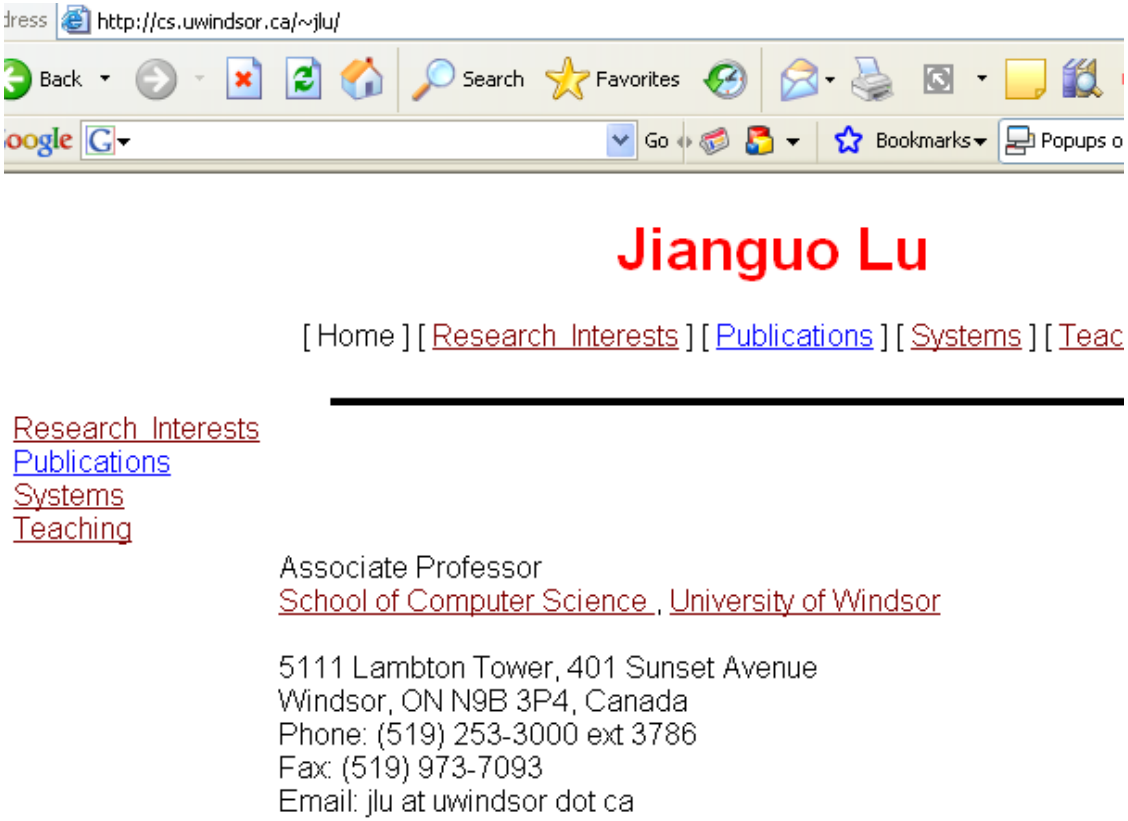
Displayed in browser



Create your web site

sol:~/public_html>pwd
/global/fac2/jlu/public_html

sol:~/public_html>ls index.html
index.html

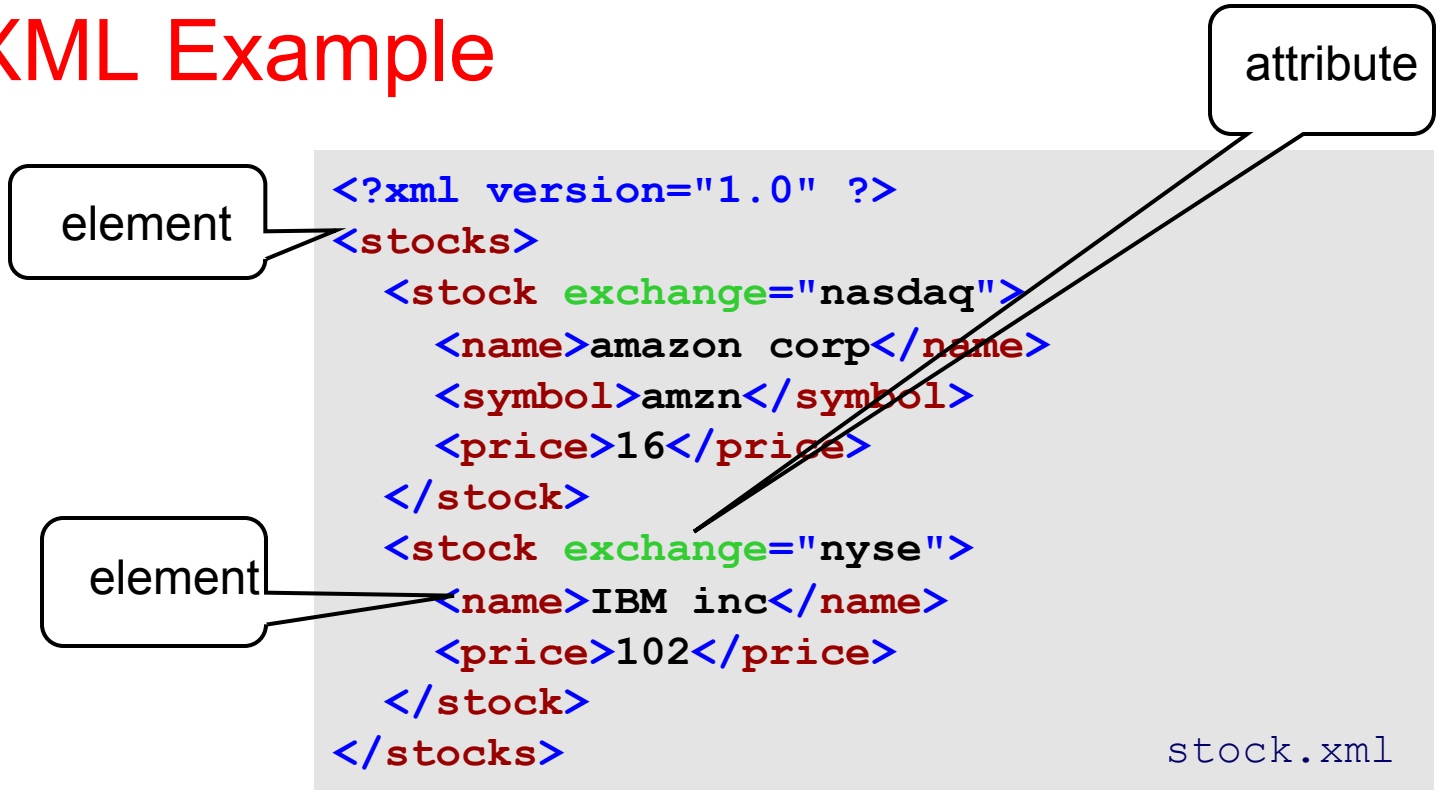


XML and HTML

- Similarities:
  - They are both markup languages;
  - They are both simple.
- Differences:

	HTML	XML
Syntax	A fixed set of tags (e.g., <table> <tr>)	Can define new tags
	Not case sensitive (e.g., <html> and <HTML> mean the same thing)	Case sensitive, e.g., <HTML> vs. <html> are different tags
	Tolerate errors	Strict syntax
	… …	
Semantics	Describe formatting info, and hyperlinks.	Describe meaning and structure of data
Pragmatics	For humans to read	For program to process
	Displayed in browsers	To be exchanged and integrated in applications

XML Example



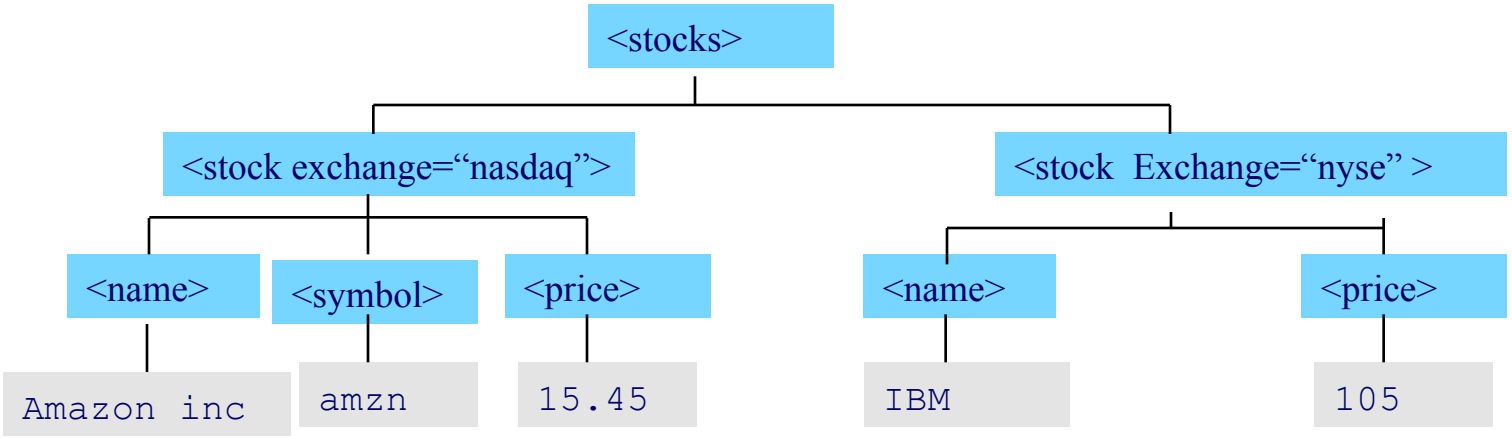
- An XML document has a group of elements;
- Each element has an opening tag and a closing tag;
- An element can have attributes.

Benefits of using XML

```
<html><body>  Stock table
<TABLE border="1">
<TR><TH> Exchange </TH> <TH>  Name        </TH> <TH> Price  </TH> </TR>
<TR><TD> nasdaq   </TD> <TD> amazon corp </TD> <TD> 16.875 </TD> </TR>
<TR><TD> nyse     </TD> <TD> IBM inc     </TD> <TD> 102.250</TD> </TR>
</TABLE> </body></html>
```

```
<?xml version="1.0" ?>
<stocks>
  <stock exchange="nasdaq">
    <name>amazon corp</name>
    <symbol>amzn</symbol>
    <price>16</price>
  </stock>
  <stock exchange="nyse">
    <name>IBM inc</name>
    <price>102</price>
  </stock>
</stocks>
```

Tree structure of XML



XML Element

- An element consists of:
  - an opening tag
  - the content
  - a closing tag
- Example:
  - <lecturer>David Billington</lecturer>
- Tag names can be chosen almost freely.
  - The first character must be a letter, an underscore, or a colon
  - No name may begin with the string “xml” in any combination of cases
    - E.g. “Xml”, “xML”

Contents of XML Elements

- Content may be text, or other elements, or nothing

```
<lecturer>
  <name>David Billington</name>
  <phone> +61 − 7 − 3875 507 </phone>
</lecturer>

<lecturer></lecturer>
```

- If there is no content, then the element is called empty; it is abbreviated as follows:

```
<lecturer/>
```

Attributes

- An attribute is a name-value pair inside the opening tag of an element

```
<lecturer name="David Billington" phone="+61 − 7 − 3875 507"/>
```

- Example

```
<order orderNo="23456" customer="John Smith"
        date="October 15, 2002">
    <item itemNo="a528" quantity="1"/>
    <item itemNo="c817" quantity="3"/>
</order>
```

Element and Attribute

```
<order>
    <orderNo>23456</orderNo>
    <customer>John Smith</customer>
    <date>October 15, 2002</date>
    <item>
        <itemNo>a528</itemNo>
        <quantity>1</quantity>
    </item>
    <item>
        <itemNo>c817</itemNo>
        <quantity>3</quantity>
    </item>
</order>
```

```
<order orderNo="23456"
        customer="John Smith"
        date="October 15, 2002">
    <item itemNo="a528“
        quantity="1"/>
    <item itemNo="c817“
        quantity="3"/>
</order>
```

- Attributes can be replaced by elements;
- When to use elements and when attributes is a matter of taste;
- But attributes **cannot** be nested.
  - Attributes can only have simple types.

Further Components of XML Docs

- Comments
  - A piece of text that is to be ignored by parser
  - <!-- This is a comment -->
- Processing Instructions (PIs)
  - Define procedural attachments
  - <?xml-stylesheet type="text/xsl" href="stock.xsl"?>
    - This instruction tells the program, say, the browser, to use stocl.xsl to process the xml document.
    - We will see this processing instruction later in XSLT.
  - <?stylesheet type="text/css" href="mystyle.css"?>

Well formed XML Document

- An XML document is well formed if it conforms to XML syntax rules.
- Additional rules:
  - XML document must have a root element
  - Attribute values must be quoted
  - XML is case sensitive
- Try to find bugs in the following XML document:

```
<?xml version="1.0" ?>
 <stocks>
  <stock exchange="nasdaq">
    <name>amazon corp </name>
    <symbol>amzn</symbol>
    <price>16</price>
  </stock>
  <stock exchange= nyse >
    <name>IBM inc</name>
    <price>  102 </PRICE>
  </stock>
 </stocks>
```

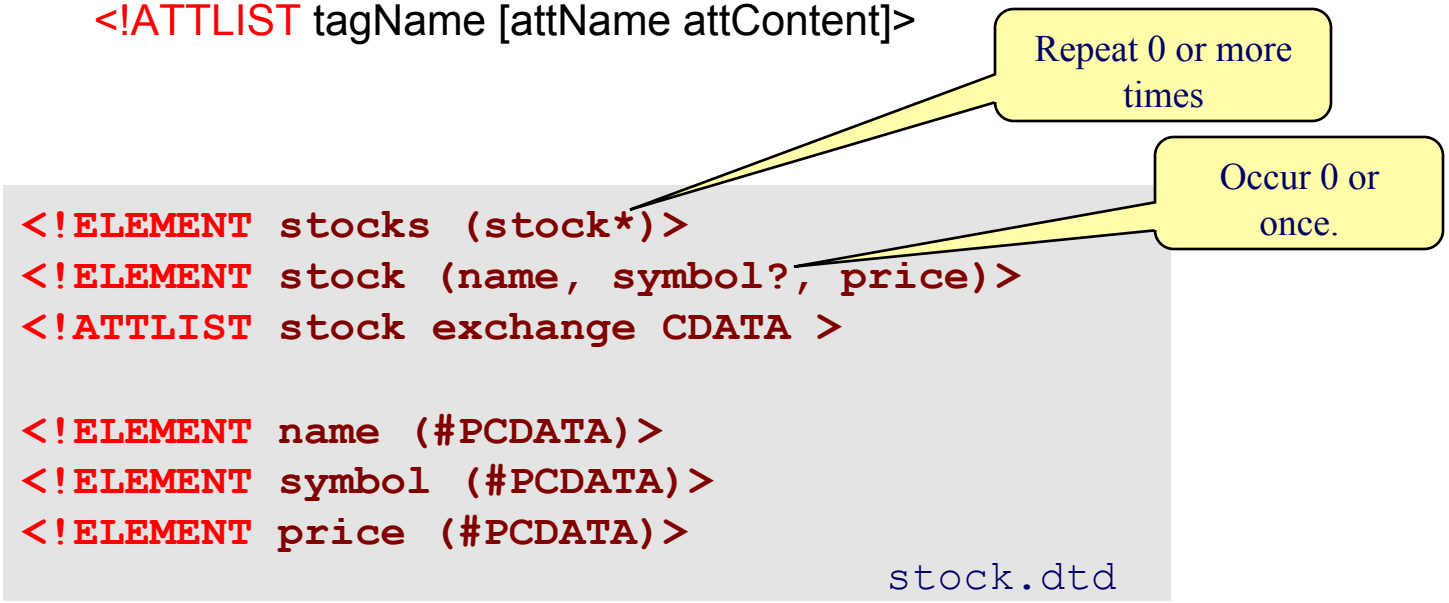
Valid XML document

```
<?xml version="1.0" ?>
<stocks>
  <name>
    <stock> 102</stock>
  </name>
  <price>IBM inc</price>
  <symbol>amzn </symbol>
  <price>16</price>
  <stock exchange="nyse">
    <price> amazon </price>
  </stock>
</stocks>
```

- Problem:
  - Not every well formed document makes sense
- Solution:
  - Associate XML with its “type”.
- Valid XML document: conforms to its DTD or XML schema.

XML DTD (Document Type Definition)

- What: DTD is a set of rules to define the syntax of a language. It is similar to context free grammar.
- Why: Help XML generation and processing.
- How: Write a sequence of element declarations and attribute declarations.
  - Element declaration
    <!ELEMENT tagName tagContent>
  - Attribute declaration
    <!ATTLIST tagName [attName attContent]>



Element Declaration

- General form:
  - <!ELEMENT tagName tagContent>
- Example:
  - <!ELEMENT stock (name, symbol?, price)>
- Content Model
  - Sequence, Choice, Cardinality
- We express that a **lecturer** element contains *either* a **name** element *or* a **phone** element as follows:

  <!ELEMENT lecturer (name | phone)>
- A **lecturer** element contains a **name** element and a **phone** element in *any order*.

  <!ELEMENT lecturer((name,phone)|(phone,name))>
- Cardinality operators
  - **?**: appears zero times or once
  - *: appears zero or more times
  - **+**: appears one or more times
  - No cardinality operator means exactly once

Attribute declaration

- General form:

  <!ATTLIST tagName [attName attContent]>

- Example:
  - <!ATTLIST stock exchange CDATA >
  - <!ATTLIST item itemNo ID #REQUIRED
    quantity CDATA #REQUIRED
    comments CDATA #IMPLIED>
- AttContent contains Attribute types and default values.

Attribute types

- Similar to predefined data types, but limited selection
- The most important types are
  - **CDATA**, a string (sequence of characters)
    - Example: <!ATTLIST stock exchange CDATA >
  - **ID**, a name that is unique across the entire XML document
  - **IDREF**, a reference to another element with an ID attribute carrying the same value as the IDREF attribute
  - **IDREFS**, a series of IDREFs
  - **(v1| . . . |vn)**, an enumeration of all possible values
- Limitations: no data types for dates, integer, number ranges etc.
  - XML Schema will solve this problem

Another DTD example

```
<order orderNo="23456“
          customer="John Smith“
          date="October 15, 2002">
  <item itemNo="a528" quantity="1"/>
  <item itemNo="c817" quantity="3"/>
</order>
```

```
<!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>
```

- ID attribute values must be unique
- IDREF attribute values must match some ID

Reference with IDREF and IDREFS

DTD:

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

What's the corresponding concepts in database ?

XML Example

```
<family>
  <person id="bob"
       mother="mary"     father="peter">
        <name>Bob Marley</name>
   </person>

  <person id="bridget" mother="mary">
        <name>Bridget Jones</name>
  </person>

   <person id="mary" children="bob bridget">
       <name>Mary Poppins</name>
   </person>

   <person id="peter" children="bob">
       <name>Peter Marley</name>
    </person>
</family>
```

Enumerated attribute values

- Syntax:

  <!ATTLIST element-name

  attribute-name (en1|en2|..) default-value>

- DTD example:

  <!ATTLIST payment method (check|cash) "cash">

- Valid XML example:

  <payment method="check" />

  or

  <payment method="cash" />

Attribute defaults

- #REQUIRED
  - Attribute must appear in every occurrence of the element type in the XML document
- #IMPLIED
  - The appearance of the attribute is optional
- #FIXED "value"
  - Every element must have this attribute value
- "value"
  - This specifies the default value for the attribute

#FIXED value

- Syntax

  <!ATTLIST element-name

  attribute-name attribute-type #FIXED "value">

- DTD example:

  <!ATTLIST sender company CDATA #FIXED "Microsoft">

- A valid XML:

  <sender company="Microsoft" />

- An invalid XML:

  <sender company=“IBM" />

Default value

- Example DTD:
  - <!ELEMENT square EMPTY>
  - <!ATTLIST square width CDATA "0">
- Valid XML:
  - <square width="100" />
  - <square/>

DTD example for email

- A **head** element contains (in that order):
  - a **from** element
  - at least one **to** element
  - zero or more **cc** elements
  - a **subject** element
- In **from**, **to**, and **cc** elements
  - the **name** attribute is not required
  - the **address** attribute is always required
- A **body** element contains
  - a **text** element
  - possibly followed by a number of **attachment** elements
- The **encoding** attribute of an **attachment** element must have either the value "**mime**" or "**binhex**"
  - "**mime**" is the default value

Email DTD

```
<!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>
```

Remarks on DTD

- A DTD can be interpreted as an Extended Backus-Naur Form (EBNF)
  - <!ELEMENT email (head, body)>
  - is equivalent to email ::= head body
- Recursive definitions possible in DTDs
  - <!ELEMENT bintree ((bintree root bintree) | emptytree)>

Where we are

- XML
- DTD
- **XML Schema**

Motivation of XML Schema

- People are dissatisfied with DTDs
  - It's a different syntax
    - You write your XML (instance) document using one syntax and the DTD using another syntax --> bad, inconsistent
  - Limited datatype capability
    - DTDs support a very limited capability for specifying datatypes. You can't, for example, express "I want the <elevation> element to hold an integer with a range of 0 to 12,000"
    - Desire a set of datatypes compatible with those found in databases
      - DTD supports 10 datatypes; XML Schemas supports 44+ datatypes

XML Schema—compared with DTD

- Specify:
  - the *structure* of instance documents
    - "this element contains these elements, which contains these other elements, etc"
  - the *datatype* of each element/attribute
    - "this element shall hold an integer with the range 0 to 12,000" (DTDs don't do too well with specifying datatypes like this)
- Significantly richer language for defining the structure of XML documents
- Its syntax is based on XML itself
  - not necessary to write separate tools
- Reuse and refinement of schemas
  - Expand or delete already existent schemas
- Sophisticated set of data types, compared to DTDs
  - Define new data types.
  - Built in data types: DTD supports 10; XML Schemas supports 44+ datatypes.
- XSD 1.0 was recommended by W3C in 2001

Some slides are from Roger Costello

From DTD to XML Schema

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Author" type="xsd:string"/>
  <xsd:element name="Date" type="xsd:string"/>
  <xsd:element name="ISBN" type="xsd:string"/>
  <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

<!ELEMENT BookStore (Book+)>

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

<!ELEMENT Title (#PCDATA)>

<!ELEMENT Author (#PCDATA)>

<!ELEMENT Date (#PCDATA)>

<!ELEMENT ISBN (#PCDATA)>

<!ELEMENT Publisher (#PCDATA)>

XML Schema syntax

- An XML schema is an element with an opening tag like

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

- Structure of schema elements
  - Element and attribute types using data types

Element Types

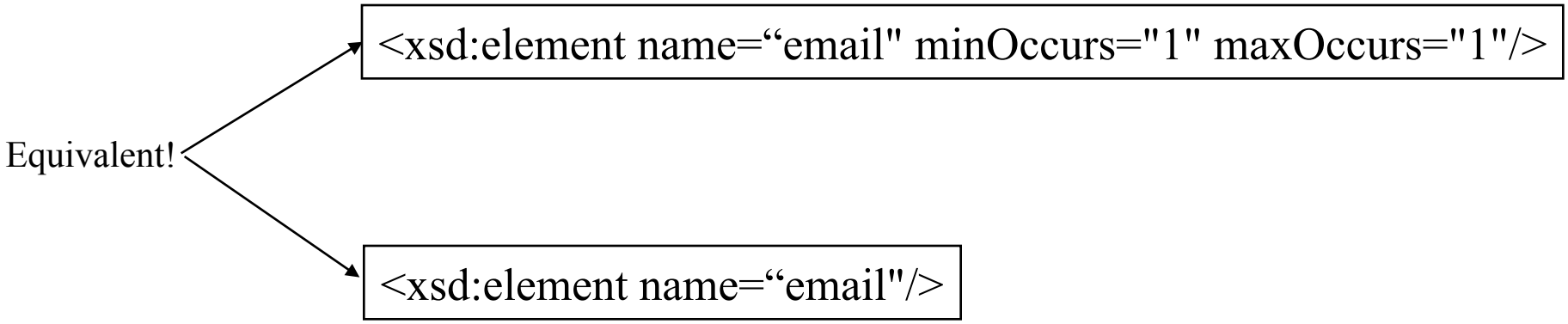
- Example

  <element name="email"/>

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

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

- Cardinality constraints:
  - minOccurs="x" (default value 1)
  - maxOccurs="x" (default value 1)
  - Generalizations of *, ?, + offered by DTDs



Attribute declaration

- A simple attribute declaration example
  - Attribute definition:

    <xs:attribute name="lang" type="xs:string"/>
  - XML example with the attribute:

    <lastname lang="EN">Smith</lastname>

- Declare default/optional/required/fixed values

  <attribute name="lang" type="xs:string" use="optional"/>

  <attribute name="lang" type="xs:string" use="required"/>
- Note that we don’t need to specify the element name as in DTD.

Data types

- Built-in data types
  - Numerical data types: integer, Short etc.
  - String types: string, ID, IDREF, CDATA etc.
  - Date and time data types: time, Month etc.
- User defined data types
  - <simpleType>: no elements or attributes are used
  - <complexType>: elements and attributes are used
- Complex data types
  - *sequence*, a sequence of existing data type elements (order is important)
  - *all*, a collection of elements that must appear (order is not important)
  - *choice*, a collection of elements, of which one will be chosen

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>
```

Data Type Extension

- Data types can be extended by new elements or attributes. Example:

```
<complexType name="extendedLecturerType">
<extension base="lecturerType">
   <sequence>
      <element name="email“   type="string“
                  minOccurs="0"  maxOccurs="1"/>
   </sequence>
   <attribute name="rank" type="string" use="required"/>
</extension>
</complexType>
```

Resulting XML Schema

```
<complexType name="extendedLecturerType">
   <sequence>
     <element name="firstname" type="string"
       minOccurs="0" maxOccurs="unbounded"/>
     <element name="lastname" type="string"/>
     <element name="email" type="string"
       minOccurs="0" maxOccurs="1"/>
   </sequence>
   <attribute name="title" type="string“   use="optional"/>
   <attribute name="rank" type="string"  use="required"/>
</complexType>
```

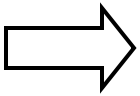
Data type restriction

- An existing data type may be restricted by adding constraints on certain values
- Restriction is not the opposite from extension
  - Restriction is not achieved by deleting elements or attributes
- The following hierarchical relationship holds:
  - Instances of the restricted type are also instances of the original type;
  - They satisfy at least the constraints of the original type.

Example of Data Type Restriction

```
<complexType name="restrictedLecturerType">
   <restriction base="lecturerType">
      <sequence>
            <element name="firstname"  type="string"
                  minOccurs="1"  maxOccurs="2"/>
      </sequence>
      <attribute name="title"  type="string"   use="required"/>
      </restriction>
</complexType>
```

Compare with the lecturerType



```
<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>
```

Restriction of simple data types

```
<simpleType name="dayOfMonth">
      <restriction base="integer">
        <minInclusive value="1"/>
        <maxInclusive value="31"/>
      </restriction>
</simpleType>

<simpleType name="dayOfWeek">
      <restriction base="string">
        <enumeration value="Mon"/>
        <enumeration value="Tue"/>
        <enumeration value="Wed"/>
        <enumeration value="Thu"/>
        <enumeration value="Fri"/>
        <enumeration value="Sat"/>
        <enumeration value="Sun"/>
      </restriction>
</simpleType>
```

Regular expression can be used

```
<xsd:simpleType name="TelephoneNumber">
   <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
      <xsd:pattern value="\d{3}-\d{4}"/>
   </xsd:restriction>
</xsd:simpleType>
```

The email example revisited

```
<element name="email" type="emailType"/>
<complexType name="emailType">
   <sequence>
      <element name="head"  type="headType"/>
      <element name="body"  type="bodyType"/>
   </sequence>
</complexType>

<complexType name="headType">
   <sequence>
      <element name="from"   type="nameAddress"/>
      <element name="to"   type="nameAddress"
        minOccurs="1"  maxOccurs="unbounded"/>
      <element  name="cc“   type="nameAddress"
        minOccurs="0"   maxOccurs="unbounded"/>
      <element  name="subject“   type="string"/>
   </sequence>
</complexType>
```

Different ways to declare elements

1

```
<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int"/>
```

2

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:complexType>
        …
    </xsd:complexType>
</xsd:element>
```

3

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:simpleType>
        <xsd:restriction base="type">
        …
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

Another way to define email schema

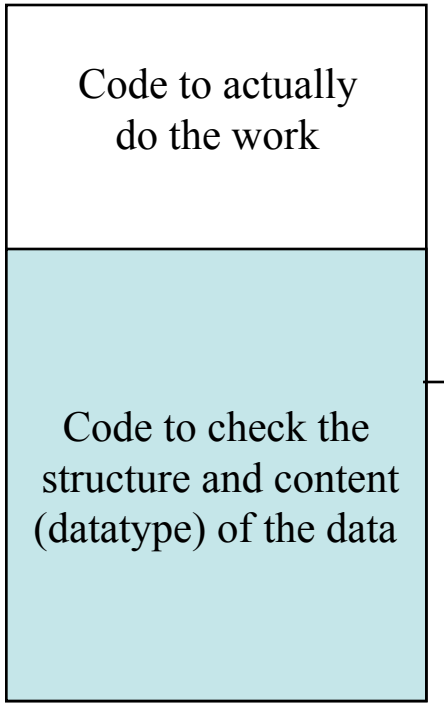
```
<element name="email">
  <complexType>
    <sequence>
      <element name="head"  type="headType"/>
      <element name="body"  type="bodyType"/>
    </sequence>
  </complexType>
</element>

<complexType name="headType">
  <sequence>
    <element name="from"   type="nameAddress"/>
    <element name="to"   type="nameAddress"
      minOccurs="1"  maxOccurs="unbounded"/>
    <element  name="cc“   type="nameAddress"
      minOccurs="0"   maxOccurs="unbounded"/>
    <element  name="subject“   type="string"/>
  </sequence>
</complexType>
```

Using XML Schema/DTD

- Data model
  - With XML Schemas you specify how your XML data will be organized, and the datatypes of your data. That is, with XML Schemas you model how your data is to be represented in an instance document.
- A contract
  - Organizations agree to structure their XML documents in conformance with an XML Schema. Thus, the XML Schema acts as a contract between the organizations.
- A rich source of metadata
  - An XML Schema document contains lots of data about the data in the XML instance documents, such as the datatype of the data, the data's range of values, how the data is related to another piece of data (parent/child, sibling relationship), i.e., XML Schemas contain metadata

Save coding

"In a typical program, up to 60% of the code is spent checking the data!“- source unknown

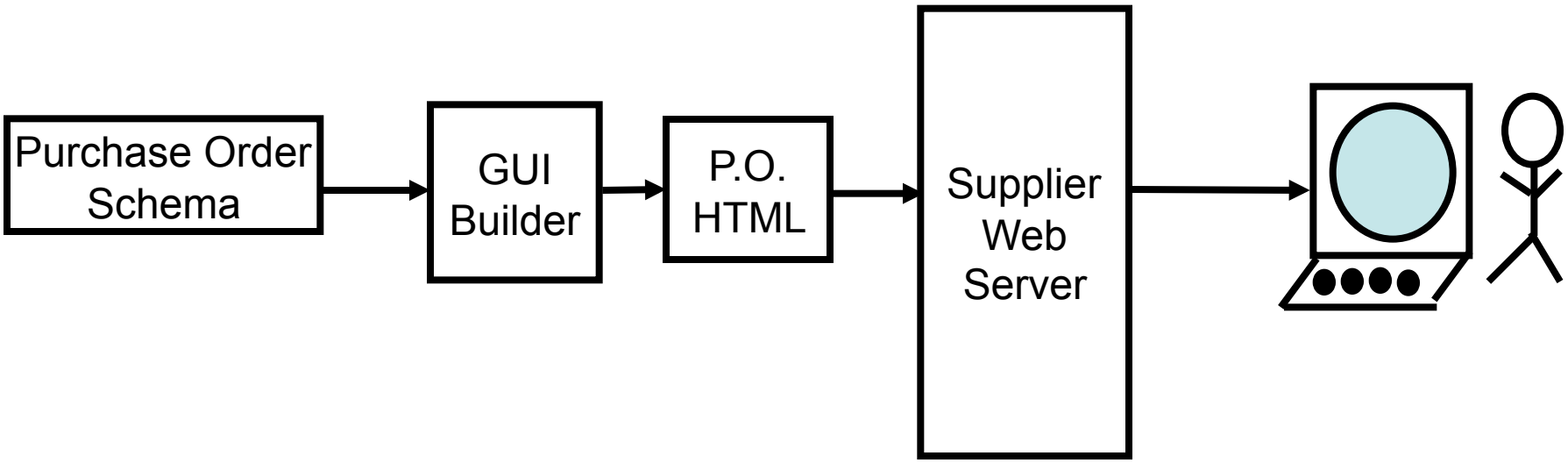


If your data is structured as XML, and there is a schema, then you can hand the data-checking task off to a schema validator.

Thus, your code is reduced by up to 60%!!!

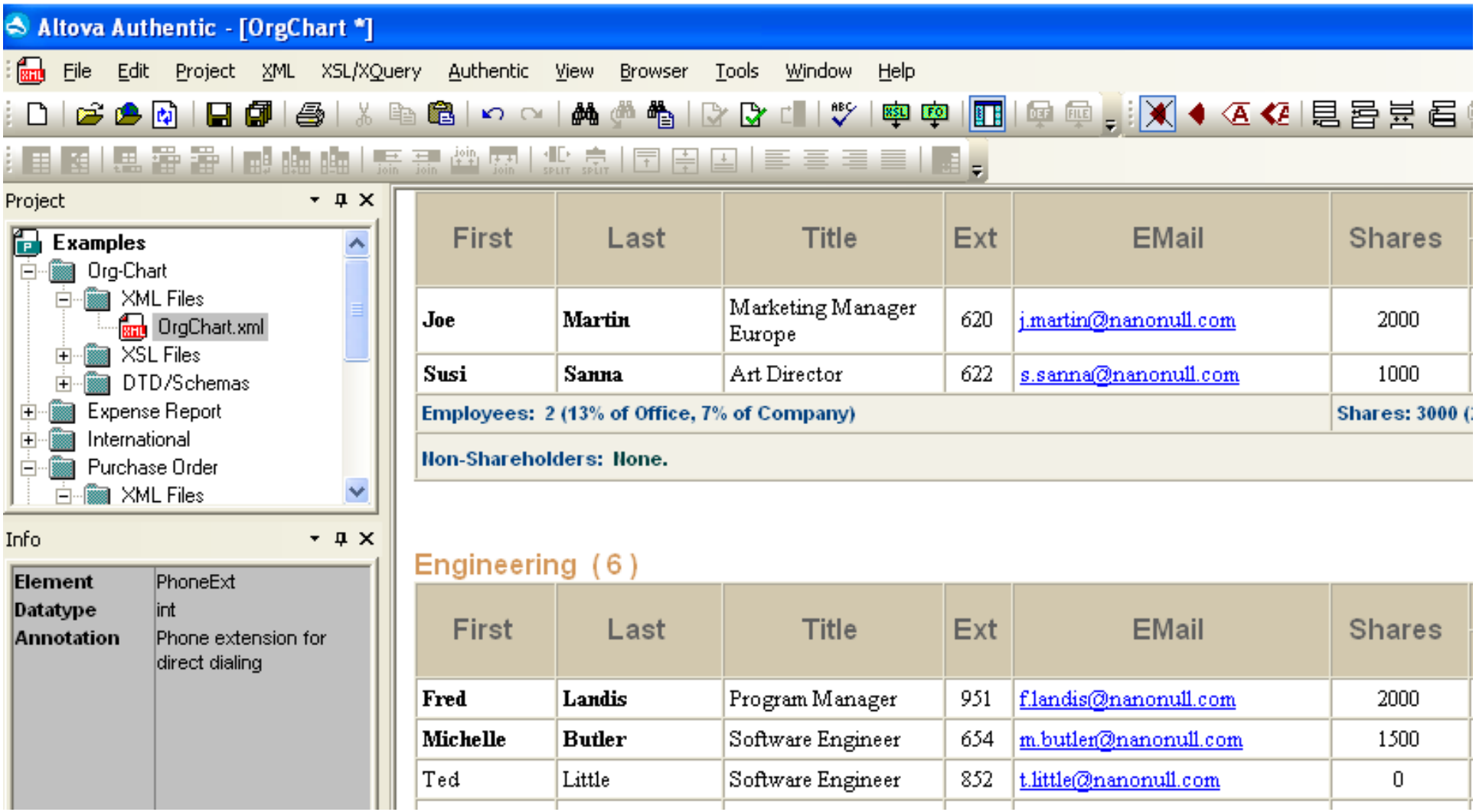
Big $$ savings!

XML-Schema to GUI

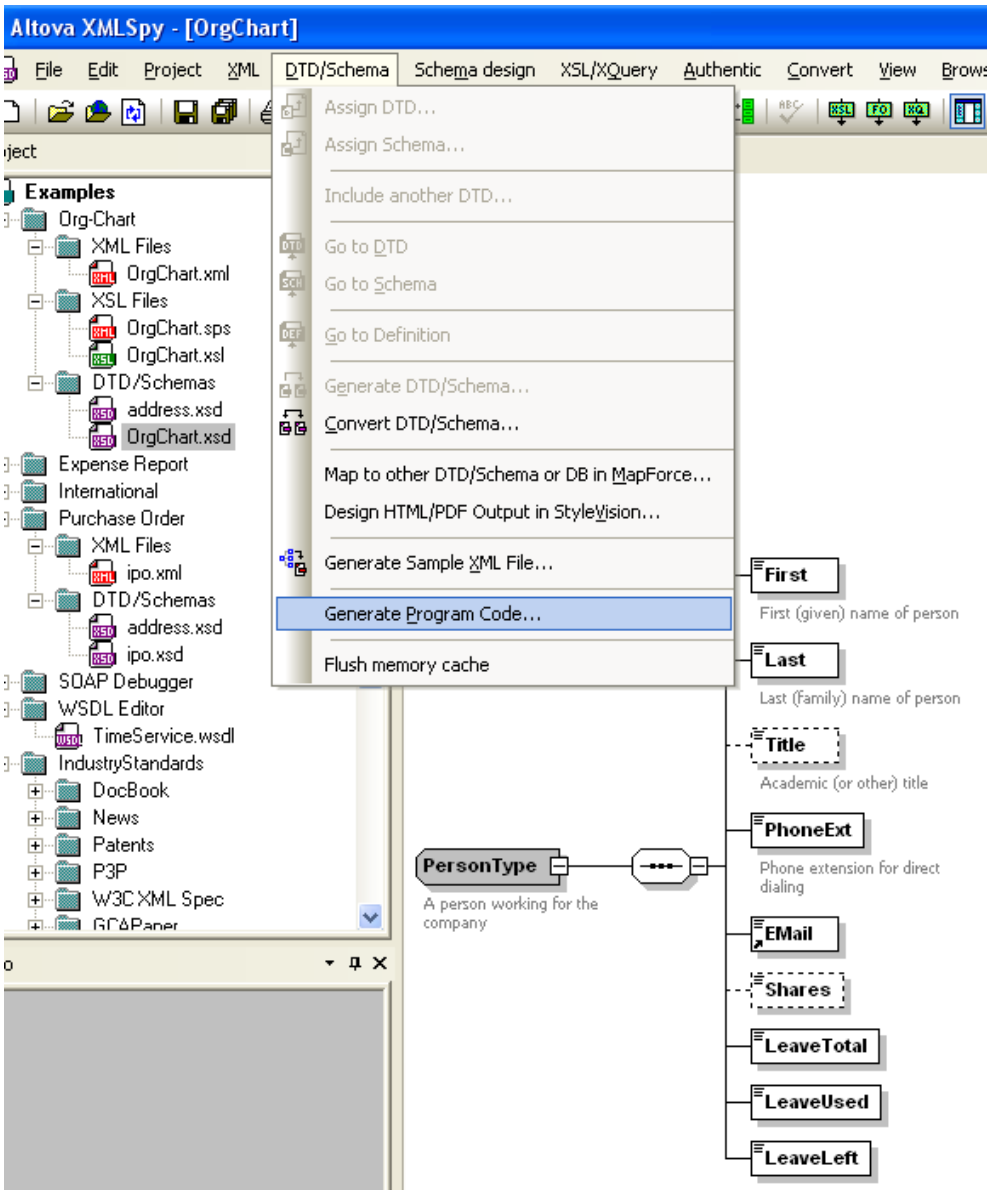


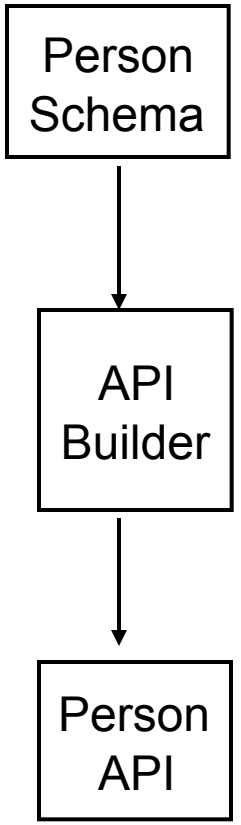
From Costello

Schema to GUI



XML Schema to API





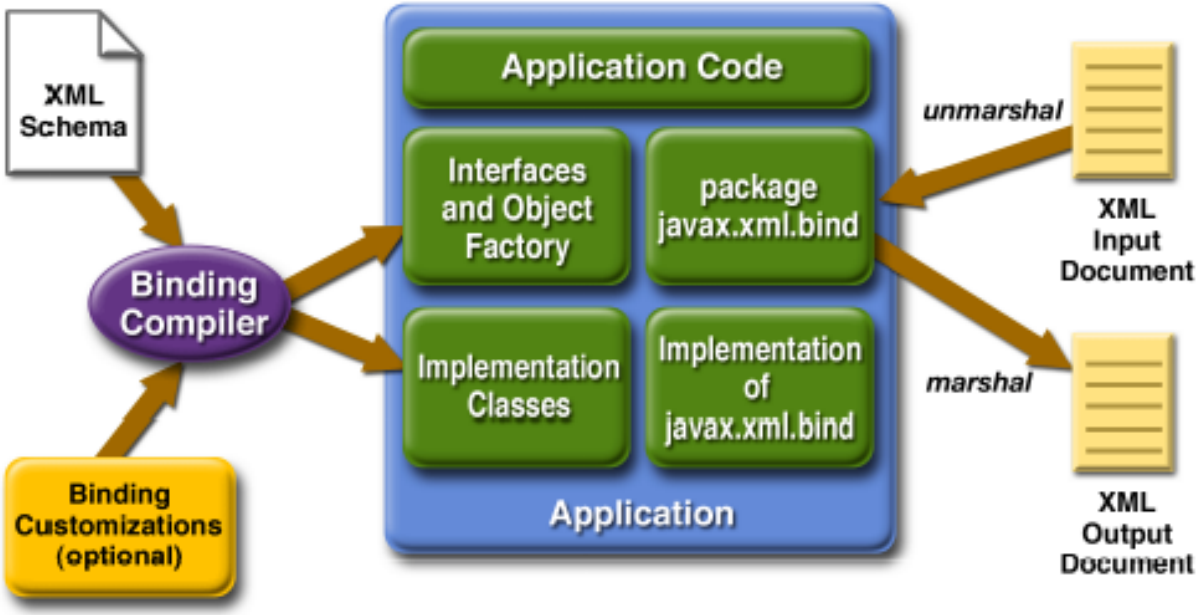
XML Schema to class

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="orders">
        <xs:sequence>
            <xs:element name="order" type="order"
    maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="order">
       <xs:sequence>
          <xs:element name="item" type="item" maxOccurs="unbounded"/
    >
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"/>
        <xs:attribute name="zip" type="xs:int" use="required"/>
    </xs:complexType>
    <xs:complexType name="item">
        <xs:sequence>
                <xs:element name="price" type="xs:double" />
                <xs:element name="quantity" type="xs:int" />
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>
</xs:schema>
```

```
public class orders {
    public order[] order;
}
public class order {
    public item[] item;
    public string id ;
    public int zip ;
}

public class item {
    public double price;
    public int quantity ;
    public string id ;
}
```

JAXB(Java Architecture for XML Binding)

- provide a fast and convenient way to bind XML schemas to Java representations,
- make it easy for Java developers to incorporate XML data and processing functions in Java applications.



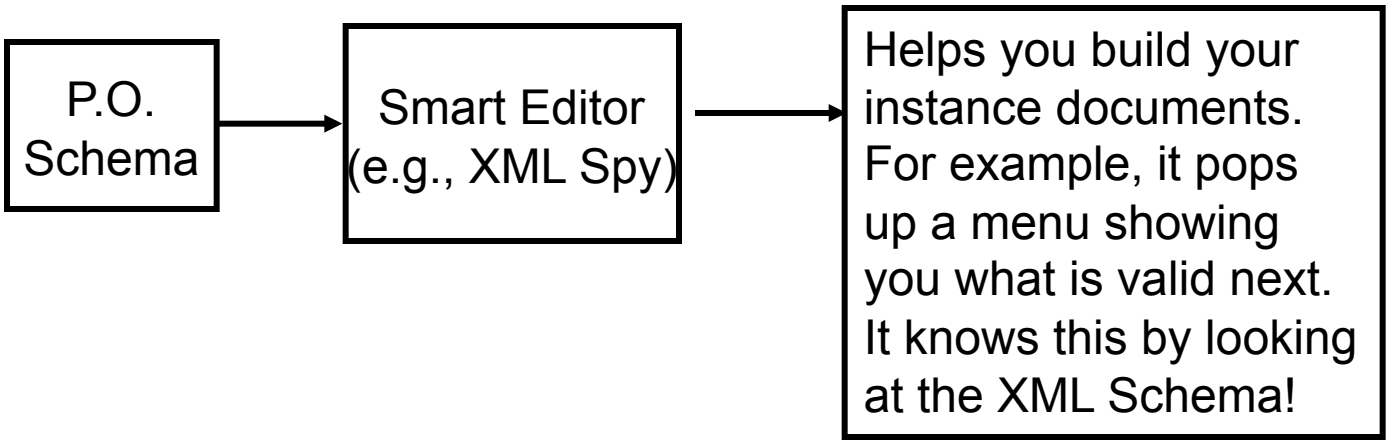
JAXB: Mapping of XML Schema Built-in Data Types

- **XML Schema Type Java Data Type**
  - xsd:string java.lang.String
  - xsd:integer java.math.BigInteger
  - xsd:int int
  - xsd.long long
  - xsd:short short
  - xsd:decimal java.math.BigDecimal
  - xsd:float float
  - xsd:double double
  - xsd:boolean boolean
  - xsd:byte byte
  - xsd:QName javax.xml.namespace.QName
  - xsd:base64Binary byte[]
  - xsd:hexBinary byte[]
  - xsd:unsignedInt long
  - xsd:unsignedShort int
  - xsd:unsignedByte short
  - xsd:time java.util.Calendar
  - xsd:date java.util.Calendar
  - xsd:anySimpleType java.lang.String

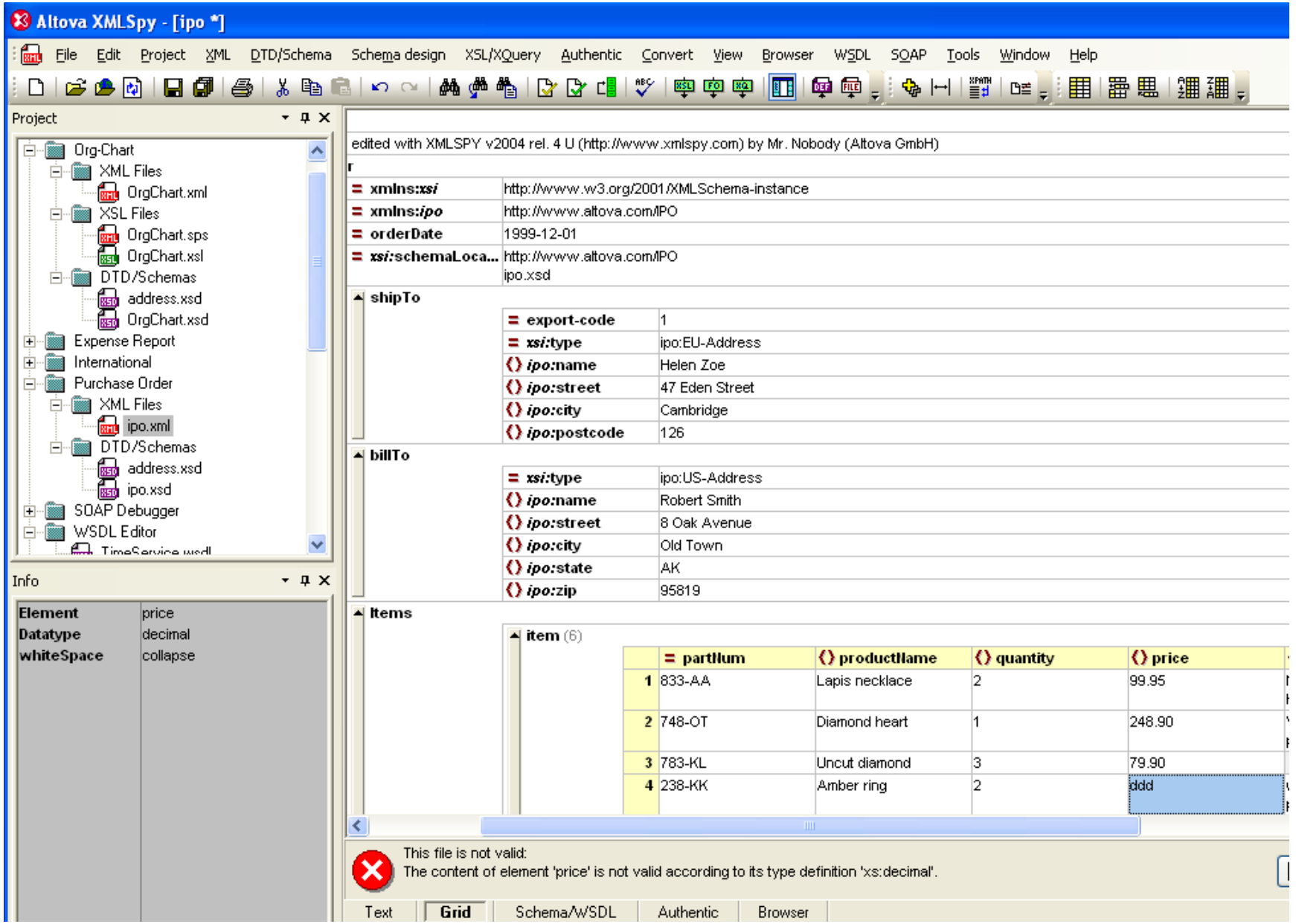
JAXB binding

- Bind the following to Java package:
  - XML Namespace URI
- Bind the following XML Schema components to Java content interface
  - Named complex type
  - Anonymous inlined type definition of an element declaration
- Bind to typesafe enum class:
  - A named simple type definition with a basetype that derives from “xsd:NCName” and has enumeration facets.
- Bind the following XML Schema components to a Java Element interface:
  - A global element declaration to a Element interface.
  - Local element declaration that can be inserted into a general content list.
- Bind to Java property:
  - Attribute use
  - Particle with a term that is an element reference or local element declaration.

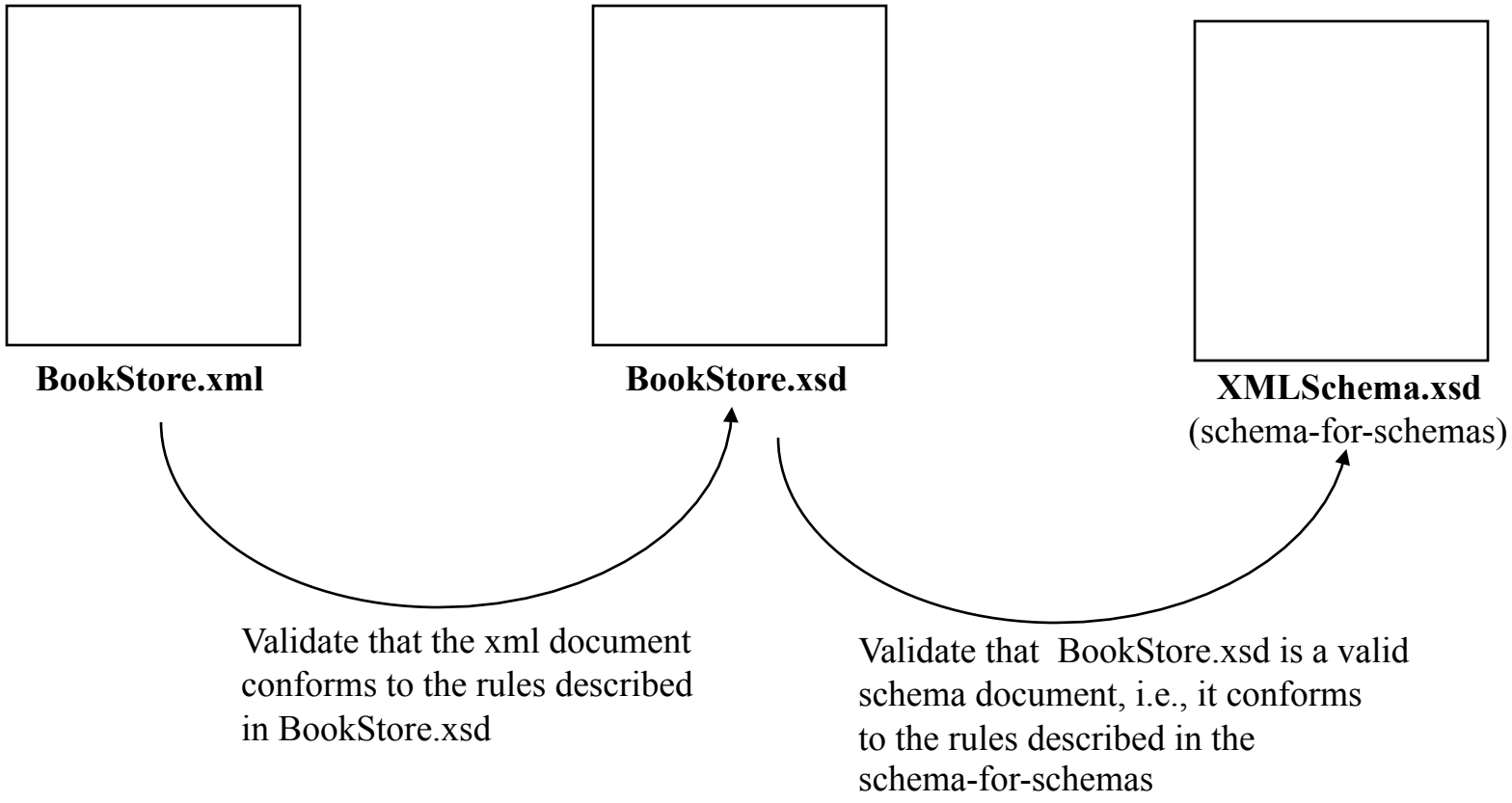
XML Schema to smart editor



XML Schema directed editor



Multiple levels of checking

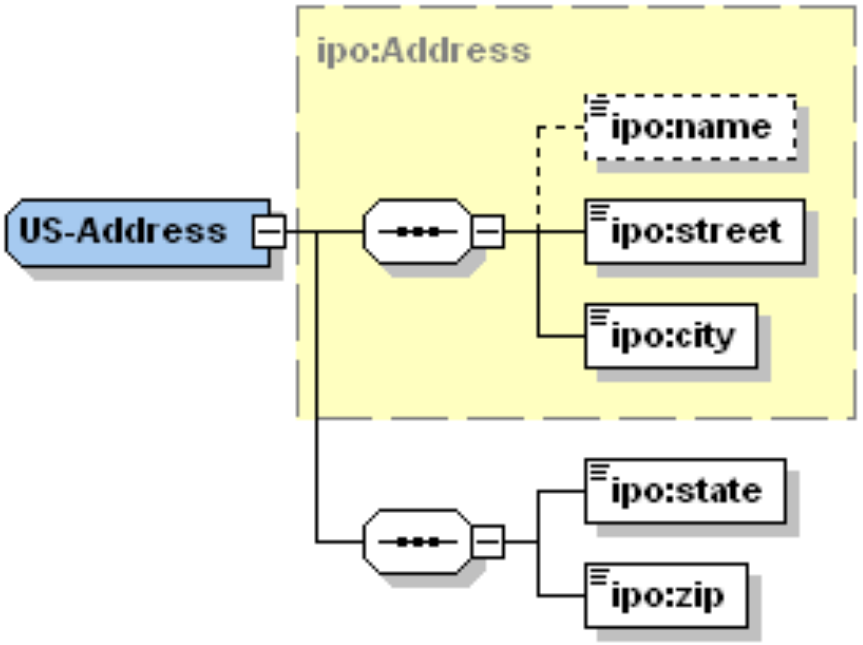


From Costello

XML Schema validators

- Command Line Only
  - XSV by Henry Thompson
    - ftp://ftp.cogsci.ed.ac.uk/pub/XSV/XSV12.EXE
- Has a Programmatic API
  - xerces by Apache
    - http://www.apache.org/xerces-j/index.html
  - IBM Schema Quality Checker (Note: this tool is only used to check your schema. It cannot be used to validate an instance document against a schema.)
    - http://www.alphaworks.ibm.com/tech/xmlsqc
  - MSXML4.0
    - http://www.microsoft.com
- GUI Oriented
  - XML Spy
    - www.altova.com/ (previously http://www.xmlspy.com)
  - Turbo XML
    - http://www.extensibility.com

XML Schema editor



Evolution of XML Schema

- XML Schema 1.0
- XML Schema 1.1
  - W3C Candidate recommendation April 2009