XML the Enabling Technology for e-Business

406.306 Management Information Systems

Jonghun Park

jonghun@snu.ac.kr

Dept. of Industrial Engineering

Seoul National University

9/20/2007

documents

- a reproducible collection of interrelated data, carried as a unity, reproduced or communicated by means of a storage medium
- characterized by contents, structure, and presentation
- appearances: physical (non-virtual), virtual
  - document imaging systems convert physical documents into a digital representation
- essential component of BPs: usually the document flow determines the BP flow
- cf: document engineering

markup

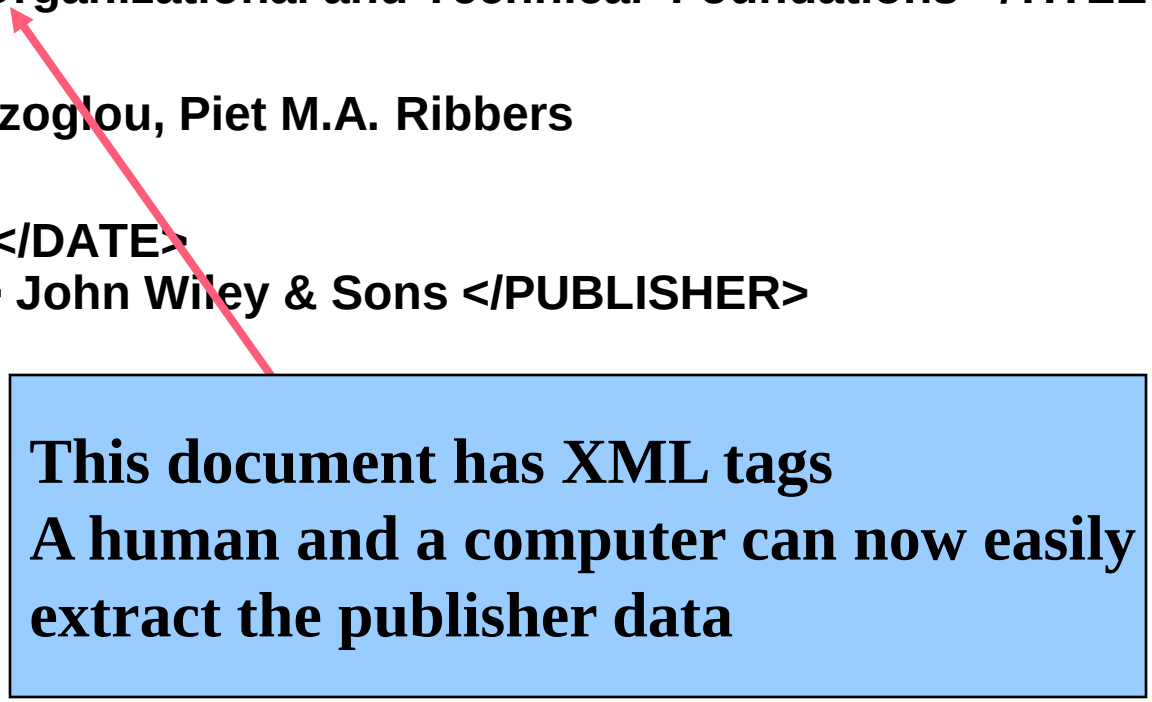
- markup
  - a method of distinguishing text from instructions in typesetting systems
  - special characters used to show when a markup instruction starts and stops
  - example: <centre on> This is a <italics on> very serious <italics off> matter.<centre off>
  - “tags” are easily understandable by both the human reader and the machine
- markup language
  - a set of conventions used together for encoding texts
  - must specify what markup is allowed, what is required, what it means, and how it is to be distinguished from text

overview of XML

- XML (eXtensible Markup Language)
- an extensible language used for the description and delivery of marked-up electronic text over the web
- a meta-language
  - provides a means of formally describing a markup language
  - allows definition of new tags and languages
  - used to structure the document, to define its contents, and to describe the data
- characteristics
  - descriptive rather than prescriptive (procedural): separation of formatting instructions (called stylesheet)
  - document type concept
  - portability

book catalog in XML

```
<BOOK>
    <TITLE>
     e-Business: Organizational and Technical  Foundations </TITLE>
    <AUTHOR>
      Mike P. Papazoglou, Piet M.A. Ribbers
    </AUTHOR>
    <DATE> 2005 </DATE>
    <PUBLISHER> John Wiley & Sons </PUBLISHER>
</BOOK>
```

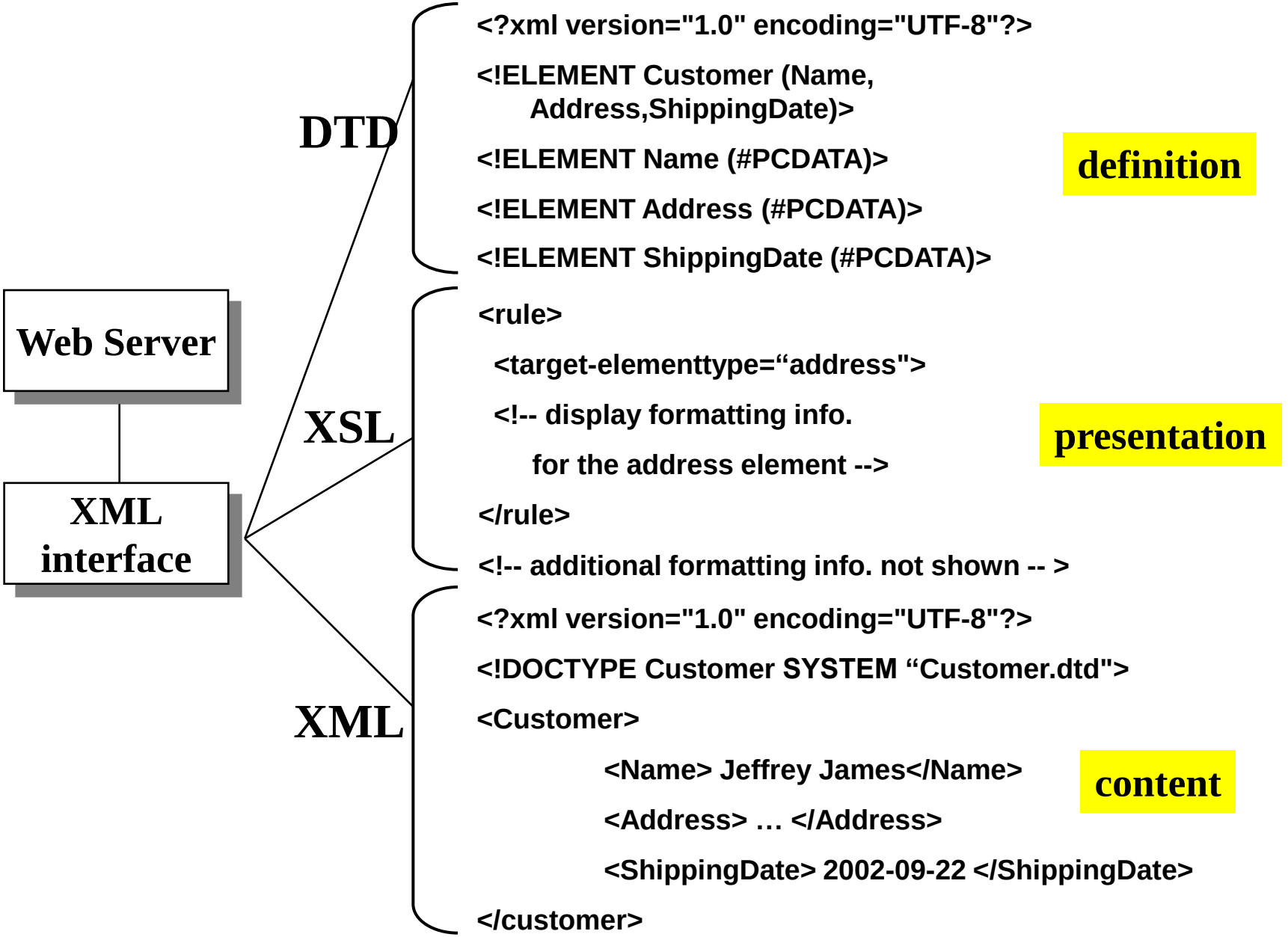


why use XML?

- simplicity
- extensibility
- interoperability
- openness
- platform-independent, industry accepted standard
- integration of all traditional databases
- one data, different views
- internationalization (Unicode)
- extensive software tools available
- industry-specific schemas

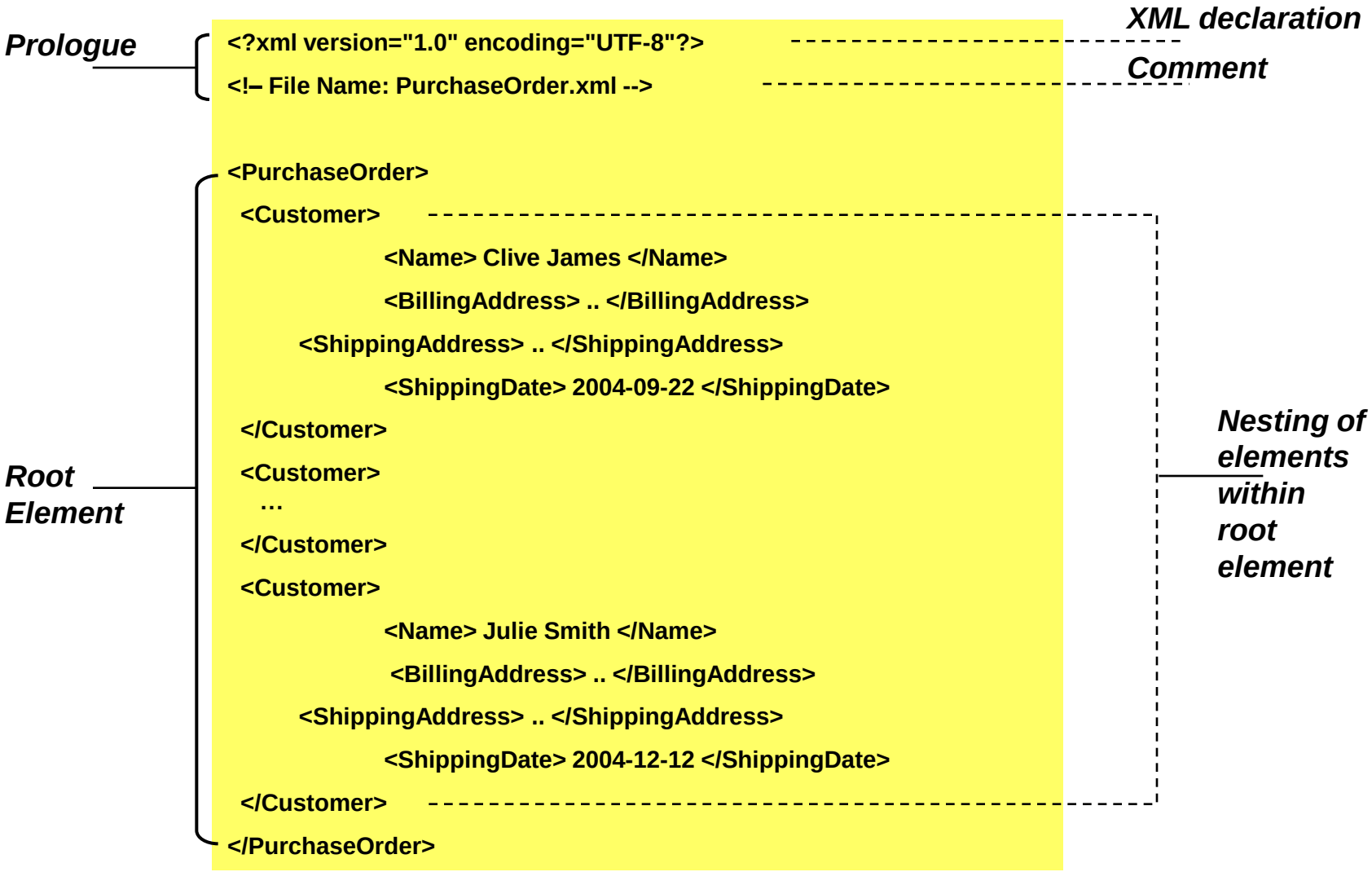
3 major XML Components

- for an XML document to be easy to be interpreted it must contain three distinct parts:
  - **content** or document data (normally text) that makes up the information contained within the document
  - definition of the **structure** and **organization** of document elements using XML DTDs or schema
  - presentation of a document's visual aspects, e.g., its style defined by means of stylesheets


Web Server
XML interface
DTD
XSL
XML
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Customer (Name, Address,ShippingDate)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT ShippingDate (#PCDATA)>
definition
<rule>
<target-elementtype="address">
<!-- display formatting info.
for the address element -->
</rule>
<!-- additional formatting info. not shown -- >
presentation
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Customer SYSTEM "Customer.dtd">
<Customer>
<Name> Jeffrey James</Name>
<Address> … </Address>
<ShippingDate> 2002-09-22 </ShippingDate>
</customer>
content

XML document

- text in XML: consists of intermingled markup and character data
- markup
  - signifies delimited text that describes the storage layout and logical structure of a document
  - all that is included between ‘<‘ and ‘>’ is considered markup, and called a tag
  - includes element start and end tags, empty element tags, comments, document type declarations, processing instructions, XML declarations, text declarations, CDATA section delimiters, entity references, character references, and any whtespace that is at the top level of the document
- consists of
  - XML declaration (prologue)
  - document element (root element)


Prologue
<?xml version="1.0" encoding="UTF-8"?>
<!– File Name: PurchaseOrder.xml -->
XML declaration
Comment
<PurchaseOrder>
<Customer>
<Name> Clive James </Name>
<BillingAddress> .. </BillingAddress>
<ShippingAddress> .. </ShippingAddress>
<ShippingDate> 2004-09-22 </ShippingDate>
</Customer>
<Customer>
…
</Customer>
<Customer>
<Name> Julie Smith </Name>
<BillingAddress> .. </BillingAddress>
<ShippingAddress> .. </ShippingAddress>
<ShippingDate> 2004-12-12 </ShippingDate>
</Customer>
</PurchaseOrder>
Root Element
Nesting of elements within root element

XML: Travel Example

a standard package request

```
<?xml version="1.0" encoding="UTF-8"?>

<LWG_PackageAvailabilityRQ xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
        instance" xsi:schemaLocation="http://www.dummy.com/XMLSpy/Submissions
        LWG_PackageAvailabilityRQ.xsd">
    <OTA_PayloadStdAttributes TimeStamp="2001-04-08T21:10:30”
            EchoToken="7656" Target="Production" Version="1"/>
    <CustomerRequest PackageType="Air Package" TravelCode="GAD20”
            CompanyCode="Cosmos Holidays" BoardCode="HB”
            BrochureCode="SummerSun">
        <DateRange StartDate="2001-10-02" Duration="P14D"/>
        <HotelReference HotelCode="ADABA"/>
        <CustomerCounts>
                <CustomerCount AgeQualifyingCode="Adult" Count="2"/>
                <CustomerCount AgeQualifyingCode="Child" Count="2"/>
                <CustomerCount AgeQualifyingCode="Infant" Count="1"/>
        </CustomerCounts>
    </CustomerRequest>
</LWG_PackageAvailabilityRQ>
```

And the response

```
<LWG_PackageAvailabilityRS xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
    instance" xsi:schemaLocation="http://www.dummy.com/XMLSpy/Submissions
    LWG_PackageAvailabilityRS.xsd">
  <OTA_PayloadStdAttributes TimeStamp="2001-04-08T21:10:30"
      EchoToken="7656" Target="Production" Version="1"/>
  <PackageSegments>
    <PackageSegment>
      <TransportInformation>
        <TravelSegment DepartureDay="Tuesday"
            DepartureDate="2001-10-02"
            DepartureTime="0500" ArrivalTime="0645"
            TravelCode="GAD20">
          <DeparturePoint>
            <Airport>AMS</Airport>
          </DeparturePoint>
          <ArrivalPoint>
            <Airport>SYDN</Airport>
          </ArrivalPoint>
        </TravelSegment>
        <Information>    etc., etc., etc.
```

elements

- indicate the logical structure of a document and contain the information content of the document
- fundamental units of content defined hierarchically
- comprised of: element name and element content
- boundary defined by start and end tags
  - <name> …. </name>
  - content of an element can be character data, other nested elements or a combination of both
- attributes
  - a name-value pair that is associated with an element
  - e.g., <customer name = “marion” shippingdate=“2007-01-01”>

XML namespaces

- to avoid name clashes
- provide a facility for associating the elements and/or attributes in all or part of a document with a particular schema
- namespace declarations have a scope: a namespace declaration is in scope for the element on which it is declared and of that element's children
- namespace name and the local name of the element together form a globally unique name known as a qualified name
  - elements not in any namespace are known as **unqualified elements**
- namespace declaration
  - xmlns attribute that identifies the values as an XML namespace
  - :prefix identifier used to identify a namespace
  - unique identifier namespaceURI: just a symbolic identifier
- 2 ways to declare a namespace
  - default namespace: declares a namespace using the xmlns attribute **without a prefix**. all descendant attributes are assumed to belong to the defined namespace
  - prefixed namespace: declares a namespace using the xmlns attribute **with a prefix**. when the prefix is associated with a particular element, then this element is assumed to belong to that namespace

```
<cust:Customer xmlns:cust= ‘urn:sample-org:Customers’>
    <name> Marion </name>
    <shippingdate > 2003-02-03  </shippingdate>
</cust:Customer>
```

unqualified elements: name, shippingdate

```
<cust:Customer xmlns:cust= ‘urn:sample-org:Customers’>
    <cust:name> Marion </cust:name>
    <cust:shippingdate> 2003-02-03 </cust:shippingdate >
</cust:Customer>
```

qualified elements: name, shippingdate

```
<Customer xmlns:cust= ‘urn:sample-org:Customers’>
    <name> Marion </name>
    < shippingdate > 2003-02-03  </ shippingdate >
<Customer>
```

qualified elements using a default namespace

well-formed XML

- well-formedness
  - every tag must have a matching end-tag
  - elements may not overlap: i.e., correctly nested
  - there must be exactly one root element
  - all attributes of an element must have different names
- not well-formed:
  <title>..</title>
  <chapter>
  <paragraph>...</chapter>
  </paragraph>
- a well-formed XML document is considered valid only if it contains proper document type declarations and if the document obyes the constratins of those declarations
- type declarations and constraints are expressed either DTD or ML schema

XML schema

- DTDs have drawbacks
  - not XML documents
  - not context-sensitive and do not support modularity
  - no fundamental element or attribute type (int, float, string) so you can not define constraints
- XML schema
  - describes the elements and attributes that may be contained in a schema-conforming document and the ways that the elements may be arranged within a document structure
- schemas benefits
  - written in XML
  - provides better constraints
  - enforce data types
  - W3C standard
- tools are available to convert DTDs to Schemas

XSDL

- provide a means for defining the **structure**, content and semantics of XML documents
- is represented as an XML document
- utilizes namespaces
- provides a built-in syntax that can be used to declare datatypes (i.e., *type* attribute)

structure of a schema definition

- the root is a *schema* element
- the *schema* element must have a declaration for the XML schema namespace
  - *xsd* and *xs* are two popular prefixes
  - the prefix is used both for **schema component elements** and in **references to built-in datatypes**
- to **validate** documents that use namespaces, you can specify a *targetNamespace* for the schema
- a component name always belongs to the target namespace
  - i.e., name="poem" instead of name="poem:poem"
- an *element* element might declare things directly, or indirectly by referencing an existing *element* declaration through the use of *ref* attributes

schema components

- subelements of schema element: i.e., element, simpleType, complexType…
- all schema components can be defined with an optional ***id* attribute** for enabling hyperlinking (e.g., XPointers)
- schema components may be extended with arbitrary attributes in any namespace other than the XML schema namespace
- any schema component may have an *annotation* element to increase readability

schema components

- data types: simple and complex and extensible data types
- element type and attribute declarations
- constraints: rules for data types and elements
- relationships: between elements
- namespaces and import/include options

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Customer SYSTEM
"Customr.dtd">
<Customer>
          <Name> Clive James </Name>
          <Address> … </Address>
          <ShippingDate>
          2002-09-22
        </ShippingDate>
</customer>
```

primitive datatypes

- common programming datatypes
  - string, boolean, decimal, float, double
- XML datatypes
  - anyURI, QName (namespace-qualified name), NOTATION (to declare that element's content conforms to a declared notation)
- binary datatypes
  - hexBinary, base64Binary
- durations
  - always starts with P (e.g., P1Y3M)
- dates, recurring dates and times
  - date, time
  - gDay, gMonth, gMonthDay

derived datatypes

- based on primitive datatypes
- XML schema datatypes spec defines some derived datatypes and provides facilities for users to define their own
- restricted numeric datatypes
  - integers: integer, positiveInteger, negativeInteger, …
  - computer word lengths: byte, short, int, …
- derived XML datatypes
  - XML attribute types: ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS
  - other XML constructs: Name, language, NCName, normalizedString, token

XML attribute types

- ID: An identifier unique to a single element in a document
- IDREF: A reference to a unique ID in the document
- IDREFS: A whitespace-separated list of IDREF values
- ENTITY: The name of an unparsed entity in the document
- ENTITIES: A whitespace-separated list of ENTITY values
- NMTOKEN: A mixture of name characters
- NMTOKENS: A whitespace-separated list of NMTOKEN values

other XML constructs

- name: a string for valid XML name
- language: en, fr, …
- NCName: No-Colon name (i.e., a name without a colon)
- normalizedString: strings in which whitespace characters such as carriage return and tab characters should be replaced by spaces
- token: strings that collapse groups of adjacent whitespace into a single space and strip leading and trailing spaces

defining user-derived datatypes

- uses *simpleType* definition element
- derivation by *list*
  - <xsd:simpleType name="dates">
    <xsd:list itemType="xsd:date"/>
    <xsd:simpleType>
- derivation by *union*
  - <xsd:simpleType name="AnyKindaDate">
    <xsd:union memberTypes="myns:Lunar xsd:date">
    </xsd:union>
    </xsd:simpleType>
- derivation by *restriction*
  - constraining facets: 12 ways that you can constrain a give datatype
  - constraining facets occur as subelements of the *restriction* element

type derivation in XSDL

- creation of a new type as a variation of an existing one
- example
  - <xsd:complexType name="internationalAddr">
    <xsd:complexContent>
    <xsd:extension base="myns:address">
    <xsd:sequence>
    <xsd:element ref="myns:countryCode"/>
    </xsd:sequence>
    </xsd:extension>
    </xsd:complexContent>
    </xsd:complexType>

constraining facets

- range restrictions
  - <xsd:simpleType name="pubYear">
    <xsd:restriction base="xsd:gYear">
    <xsd:minInclusive value="1000"/>
    <xsd:maxInclusive value="2300"/>
    </xsd:restriction>
    </xsd:simpleType>
- length restrictions: minLength, maxLength
- decimal digit restrictions: totalDigits, fractionDigits
- *whiteSpace* facet
  - constrains the procesing the whitespace characters
  - preserve, replace, collapse

enumeration facets

- <xsd:simpleType name=“workday”>
  <xsd:restriction base=“xsd:string”>
    <xsd:enumeration value=“Monday”/>
    <xsd:enumeration value=“Tuesday”/>
  <xsd:restriction>
 <xsd:simpleType>

pattern facets

- have value attribute that is a regular expression
- constructing regular expressions
  - a character is a regular expression
  - quantifier
    - +, *, ?
    - {}
    - Example:

```
<xsd:simpleType name="reg-exp">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="ab{2,5}c+d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

pattern facets (cont.)

- constructing regular expressions (cont.)
  - alternatives and grouping
    - (), |
    - Example: yes+|no+, (yes)+|(no)+
  - special characters
    - \n, \r, \t
    - The special characters defined for the regular expression can be converted to ordinary characters by using "\"
    - Example: \?, *
  - character classes
    - \d: represents digits (while \D represents anything that is not a digit)
    - \s: any whitespace character
    - \i: any initial name characters
    - \w: word characters
    - Example: \d+a\d{2,}

schema and instance document

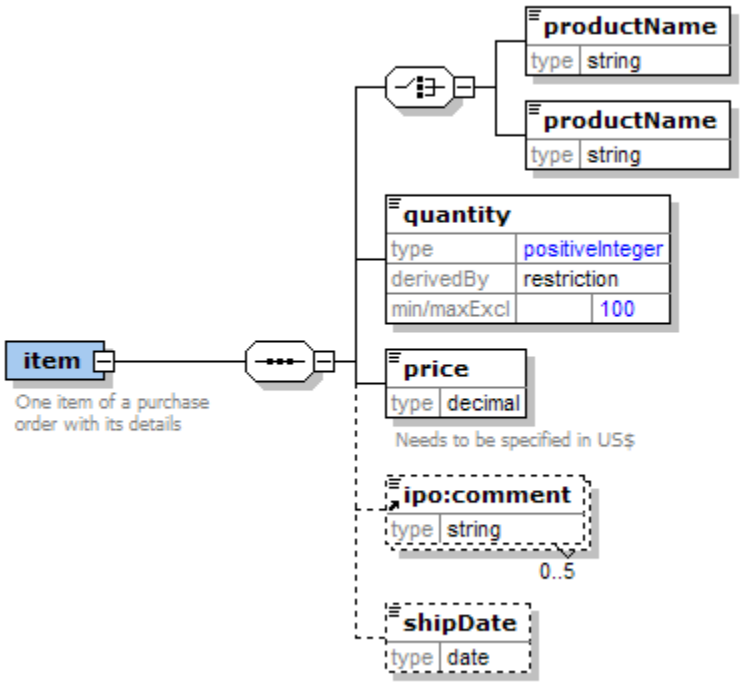
File “Person.xml”

```
<?xml version="1.0"
encoding="UTF-8"?>

<Person
xmlns:xsi="http://www.w3.org/
   2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="
Person.xsd">

      <First>Sophie</First>

      <Last>Jones</Last>

      <Age>34</Age>

</Person>
```

File “Person.xsd”

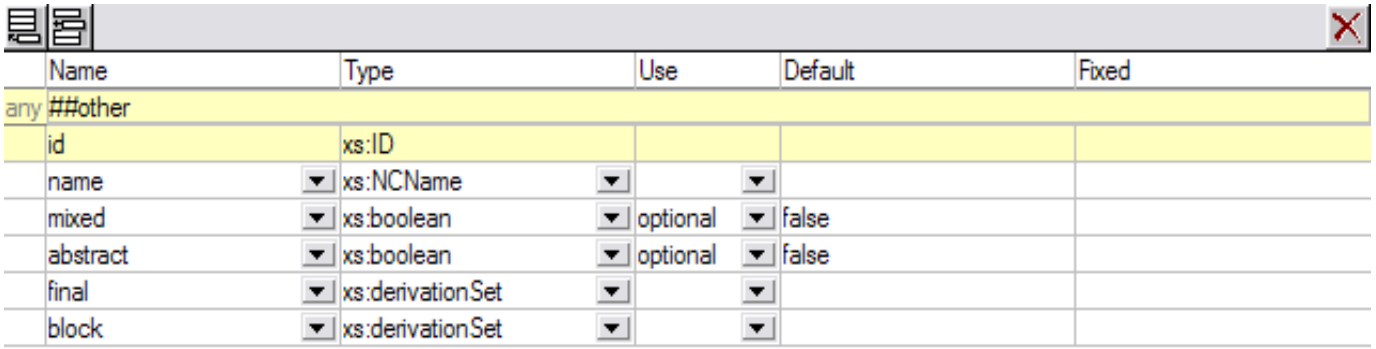
```
<?xml version="1.0" encoding="UTF-
8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/
             XMLSchema">
 <xs:element name="Person">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="First"
          type="xs:string"/>
    <xs:element name="Middle"
          type="xs:string"
          minOccurs="0"/>
    <xs:element name="Last"
          type="xs:string"/>
    <xs:element name="Age"
          type="xs:integer"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

elements and their content model




```
<element name="item">
  <annotation>
    <documentation>One item of a purchase order with its
                   details</documentation>
  </annotation>
  <complexType>
    <sequence>
      <choice>
        <element name="productName" type="string"/>
        <element name="productName" type="string"/>
      </choice>
      <element name="quantity">
        <simpleType>
          <restriction base="positiveInteger">
            <maxExclusive value="100"/>
          </restriction>
        </simpleType>
      </element>
      <element name="price" type="decimal">
        <annotation>
          <documentation>Needs to be specified in
                         US$</documentation>
        </annotation>
      </element>
      <element ref="ipo:comment" minOccurs="0" maxOccurs="5"/>
      <element name="shipDate" type="date" minOccurs="0"/>
    </sequence>
    <attribute name="partNum" type="ipo:Sku"/>
  </complexType>
</element>
```

attributes and attribute groups



	Name	Type	Use	Default	Fixed
any	##other				
	id	xs:ID			
	name	xs:NCName			
	mixed	xs:boolean	optional	false	
	abstract	xs:boolean	optional	false	
	final	xs:derivationSet			
	block	xs:derivationSet			

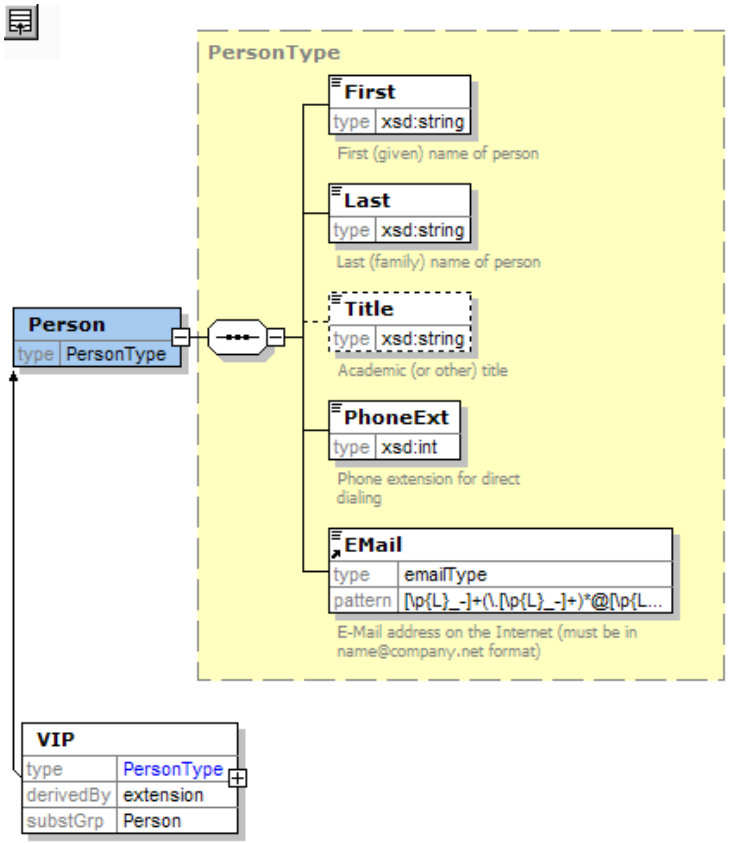
```
<xs:attribute name="name" type="xs:NCName"/>
<xs:attribute name="mixed" type="xs:boolean" use="optional" default="false"/>
<xs:attribute name="abstract" type="xs:boolean" use="optional" default="false"/>
<xs:attribute name="final" type="xs:derivationSet"/>
<xs:attribute name="block" type="xs:derivationSet"/>
```

attribute declaration in XSDL

- optional attributes
  - <xsd:attribute name="publisher" type="xsd:string"/>
  - <xsd:attribute name="pubyear" type="xsd:NMTOKEN"/>
- required attributes
  - <xsd:attribute name="href" use="required" type="xsd:anyURI" />
  - <xsd:attribute name="pubdate" use="required" type="myns:pubyear"/>
- *attribute* element in XSDL may have a default *attribute* and *fixed* attribute

complex types

```
<xsd:complexType name="PersonType">
 <xsd:sequence>
  <xsd:element name="First" type="xsd:string"/>
  <xsd:element name="Last" type="xsd:string"/>
  <xsd:element name="Title" type="xsd:string"
               minOccurs="0"/>
  <xsd:element name="PhoneExt" type="xsd:int"/>
  <xsd:element ref="EMail"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:element name="Person" type="PersonType"/>
<xsd:element name="VIP"
             substitutionGroup="Person">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="PersonType">
    <xsd:attribute name="IQ" type="xsd:int"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
```



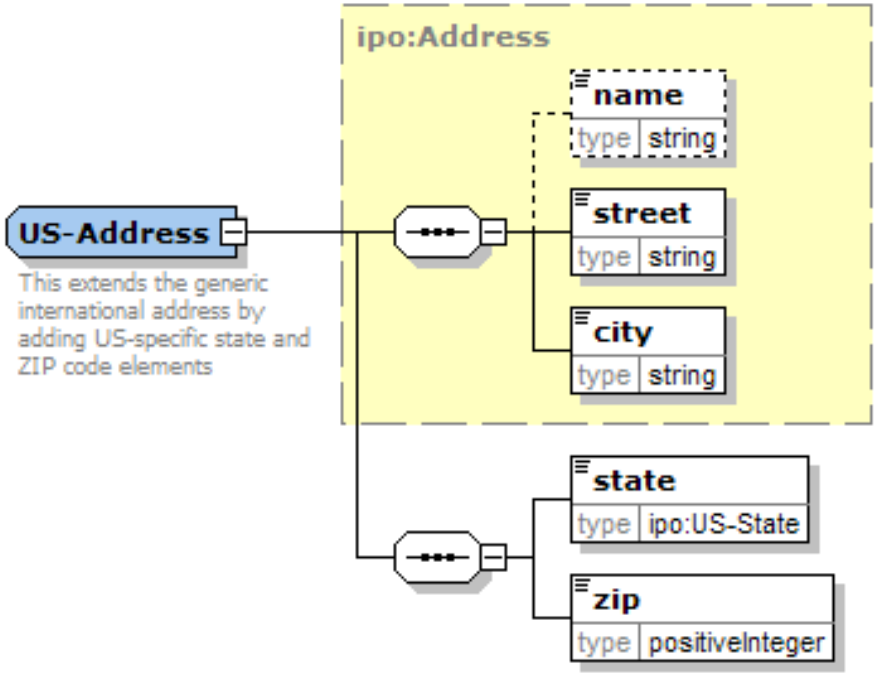
content models

- describes what content is allowed within an element
- sequence element
- choice element
- all element

```
<xsd:complexType name="complexEnough">
  <xsd:sequence>
    <xsd:element ref="myns:A"/>
    <xsd:element ref="myns:B"/>
    <xsd:choice>
      <xsd:element ref="myns:C"/>
      <xsd:element ref="myns:D"/>
    <xsd:choice>
    <xsd:all>
        <xsd:element ref="myns:E"/>
        <xsd:element ref="myns:F"/>
    </xsd:all>
  </xsd:sequence>
</xsd:complexType>
```

extension of complex type

```
<complexType name="Address">
 <sequence>
  <element name="name" type="string"
           minOccurs="0"/>
  <element name="street"
           type="string"/>
  <element name="city" type="string"/>
 </sequence>
</complexType>
<complexType name="US-Address">
 <complexContent>
  <extension base="ipo:Address">
   <sequence>
    <element name="state"
             type="ipo:US-State"/>
    <element name="zip"
             type="positiveInteger"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```



declaring schema conformance

- to allow the document author to give a more explicit hint to the receiver
- example of referring to a schema definition in a XML document
  - <myns:mydoc xmlns:myns="http://www.a.com/myns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.a.com/myns http://www.b.com/myxsdl.xsd">
  - xsi prefix: identifies a namespace to put XML schema information into the instance document
  - *schemaLocation* is an attribute defined in the xsi namespace to allow an XML document to point to an appropriate schema definition
  - http://www.a.com/myns is a namespace URI and http://www.b.com/myxsdl.xsd is a URI for the schema definition

document presentation and transformation

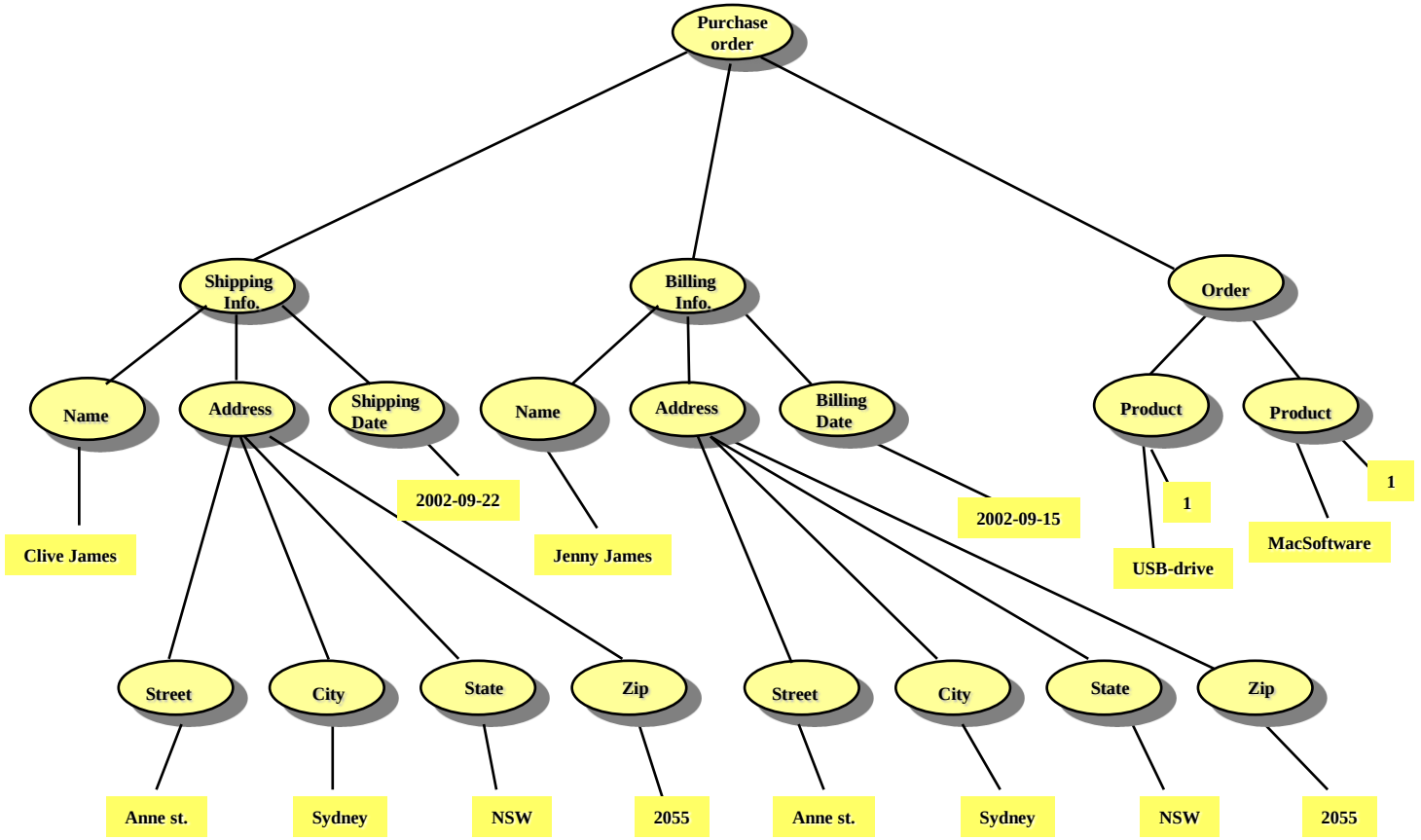
- XML presentation facilities provide a modular way to display document content on a variety of devices such as web browsers, WAP browsers, VoiceXML browsers
- XSL (eXtensible Stylesheet Language)
  - specifies how the XML data will be displayed
  - not only determine the **style** or presentation of the XML document, but also the **medium** on which it is presented, such as a browser and hand-held device
  - XSLT (XSL Transformation): transformation sublanguage
  - XSL-FO (XSL Formatting Objects)
    - formatting sublanguage
    - specifies the document styling

XSLT

- a data-driven specification that allows the conversion of an XML document into **another XML document**, or into **any other type of document** (e.g., PDF, HTML, postscript)
- XSLT style sheet contains **instructions** that tell the transformation processor how to process a source document to produce a target document
- style sheet processor creates formatted object tree
- 3 ways of XSLT transformation
  - **server side** transformation: only the resultant style sheet is displayed on the client side
  - **client side** transformation: both the source XML doc and the template are sent to the client
  - **3rd party** software transformation: both client and server see only the transformed document

XPath

- abstract language that defines a **tree model** that codifies the logical structure of an XML document against which all expressions are evaluated.



XPath and XSLT

- XPath is used to define search expressions that **locate** document elements to be used in the process of transformation.
- an XSL style sheet contains a series of **pattern-action** rules, called **templates**, each of which specifies a pattern to be matched in the source document, and a set of actions to be taken when the pattern is matched
  - a pattern is a restricted XPath location path that associates a template with a set of nodes in the source document
  - results of the actions become the body of the target document
- XSLT uses the formatting instructions in the style sheet to perform the transformation
- the converted document can be another XML document or a document in another format, such as HTML, that can be displayed on a browser
- formatting languages, such as XSLT and XSL, can access only the elements of a document that are defined by the document structure, e.g., XML schema

```
<Customer>
  <Name fname="Clive" lname="James" />
  <Address> .. </Address>
  <ShippingDate> 2002-09-22 </ShippingDate >
</Customer>
```

```
<Customer>
  <Name>
<FName> Clive </Fname>
        <LName> James </LName>
  <Name/>
  <Address> .. </Address>
</Customer>
```

Example of document transformation.

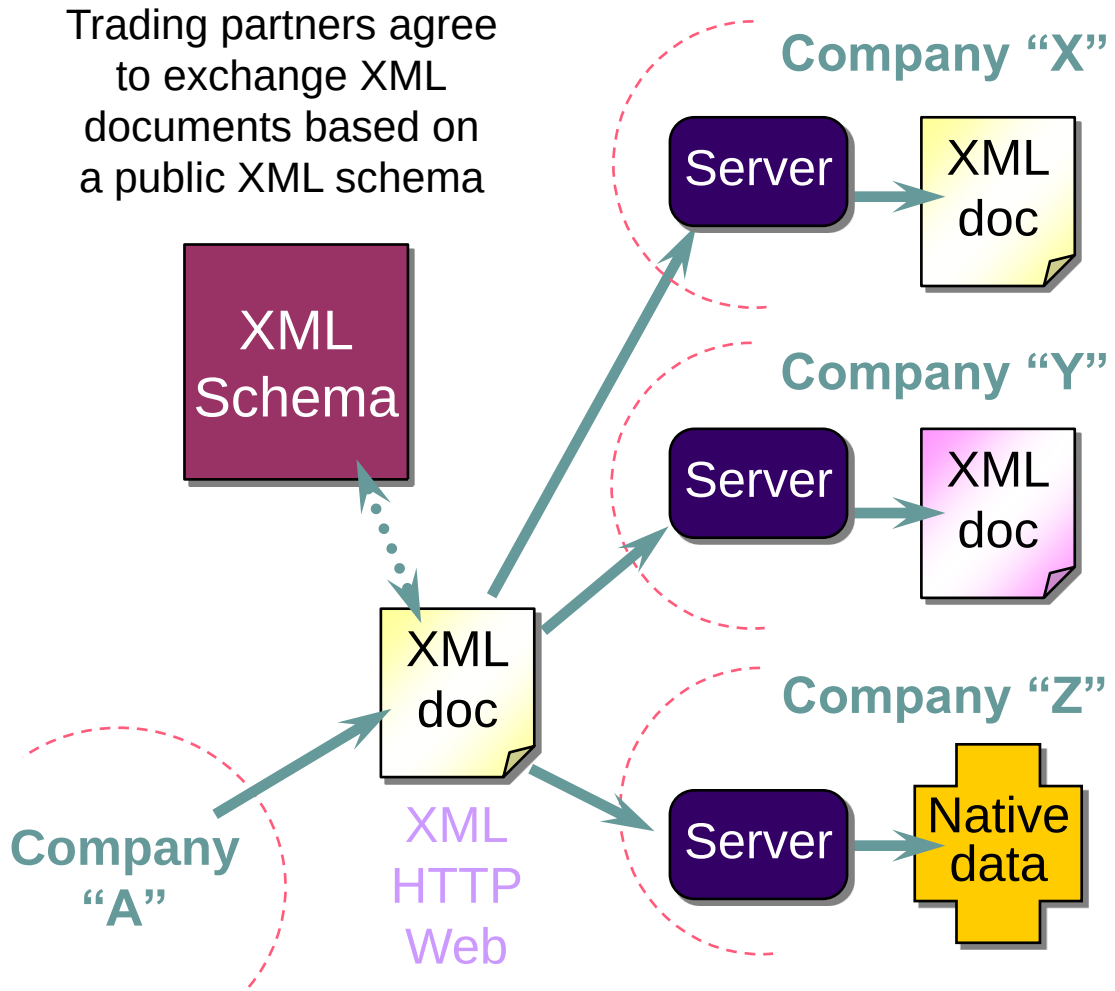
processing XML documents

- XML parsers
  - read XML documents, **validate** document syntax against a DTD or XML schema, and then **allow programmatic access** to a document's contents depending on application needs
- 2 major document parsing and processing alternatives
  - simple API for XML (SAX)
    - a set of abstract programmatic interfaces designed for the java programming language that project an XML doc onto a stream of well-known method calls
    - defines an API for an **event-based parser**
  - document object model (DOM)
    - a set of abstract programmatic interfaces that project an XML document onto a **tree structure**
    - builds a hiearchical model of the parsed XML document
    - DOM API provides a platform and language-independent interface to allow developers to programmatically access and modify the content nof tree-structured documents
  - SAX is often preferred over the DOM when performance becomes an issue

XML and its application to e-Business

- goal: to enable the creation of an ”electronic enterprise" by using XML to exchange information between the applications required to manage a set of related businesses
- ebXML
  - high-level: identifies common cross-industry business processes that characterize business transactions, and define a structure of those processes that enable development of a BPSS
  - detail-level: develops core components that addresses semantic interoperability

Scenarios for e-Business XML Document Sharing



Internally routes and processes the exact same XML document

Internally transforms XML document into a new XML document that conforms to a private XML schema

Internally transforms XML document into a non-XML, proprietary data format