ID2208 Programming Web Services

XML Overview

Mihhail Matskin:
http://people.kth.se/~misha/ID2208/index.html

Spring 2016

Content

- Introduction
  - Historical remarks
- Markups
- Data-centric and document-centric views
- XML elements
- Namespaces
- Schema
- XML processing

This lecture reference

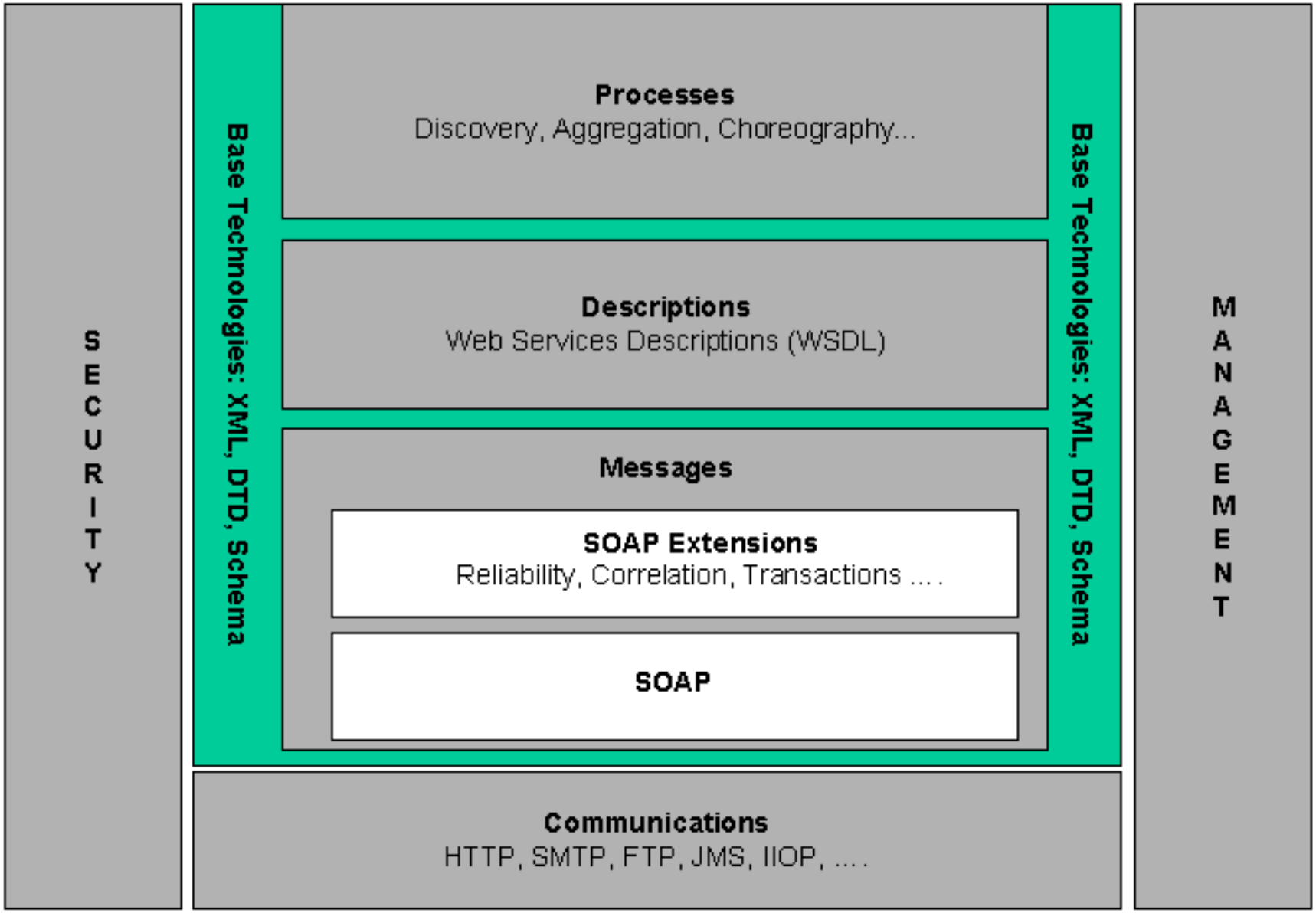
- Text book **Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2nd Edition**

Chapter 2

Literature used (recommended)

- M. P. Papazoglou. Web Services: Principles and Technology Pearson Prentice Hall, 2012
- H. M. Deitel at al. Web Services. A Technical Introduction. Pearson Education. 2003
- E. Cerami. Web Services Essentials. O'Reilly and Associates.2002.
- XML and Web Services, Sams
- G. Glass. Web Services. Building Blocks for Distributed Systems. Prentice Hall. 2002
- G. Alonso. Web Services. Concepts, Architectures and Applications. Springer, 2004

Web Service Architecture Stack



XML (eXtensible Markup Language) – in brief

- XML is simply a set of rules, that identifies how you can define tags, that separate a document into individual parts and subparts.
- Developed from Standard Generalized Markup Language (SGML)
- Defined by W3C as an open, standard technology
- Data independence – separation of content from its presentation
- Any application that understands XML has ability to format XML in a variety of different ways
- XML parser checks document's syntax (well formed)
- XML document can reference another document that defines XML document's structure Data Type Definition (DTD) or a Schema (valid)

Early beginnings –a foundation in SGML

- SGML was developed from General Markup Language (GML)
- SGML is a meta-language – it defines how any given markup language (SGML applications) can be formally specified
- Main ideas
  - system independent common way of describing data
  - tagging data as a way of identifying the structure and describing the data of the document
  - document formatting information is located separately
- SGML was too complex – it defines everything you could ever want to know about markups and more – SGML-enabled software was expensive

Early beginnings – HTML

- Developers looked toward SGML as a model for creating HTML – the most popular SGML application
- HTML combines markups from several different categories
- The HTML specification is owned by W3C
- Problem: leading browser vendors introduced a number of incompatible tags completely outside a scope of the HTML specification
- The need to simplify SGML coincided with the need to control the evolution of HTML and create a simple generalized markup for use on the Web

XML

- Then with explosion of Internet, SGML got a new “lease of life” in the form of XML – a “lightweight” version of SGML
- The XML industry experienced a boom a few years ago
- XML was suggested as the de facto standard for representing structured and semi-structured information in textual form

The Main Difference Between XML and HTML

- XML was designed to carry data.
- XML is not a replacement for HTML.
  XML and HTML were designed with different goals:
  - XML was designed to describe data and to focus on what data is.
  - HTML was designed to display data and to focus on how data looks.
- HTML is about displaying information, while XML is about describing information.
- XML is Free and Extensible
- Tags in XML are not predefined

XML Basics (markups)

- An XML document contains only text

- Data is marked up using tags:

```
<person>
      Alan Turing
</person>
```

Why markups?

- Humans can read it
- It is also quite easy to process with software
- This is inherently extensible way of representing information

Document-Centric XML

```
<H1>Skateboard Usage Requirements</H1>
<P>In order to use the <B>FastGlide</B>
  skateboard you have to have:</P>
<LIST>
  <ITEM> A strong pair of legs.</ITEM>
  <ITEM> A reasonable long stretch of smooth
  road surface.</ITEM>
  <ITEM> The impulse to impress others.</ITEM>
<P>If you have all of the above, you can
  proceed to <LINK HREF="Chapter2.xml">Getting
  on the Board</LINK>.</P>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Data-Centric XML

```
<po id="43871" submitted="2004-06-05">
  <billTo>
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
  </billTo>
  <shipTo>
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
  </shipTo>
  <order>
    <item sku="318-BP" quantity="5">
        <description>Skateboard backpack</description>
    </item>
    <item sku="947-TI" quantity="5">
        <description> Street-style titanium skateboard.</description>
    </item>
  </order>
</po>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Data- vs. Document-Centric XML

- The ratio of markup to content in data-centric is much higher than in document-centric XML
- Data-centric often includes a machine-generated information
- The document and tags are highly structured in data-centric – compare to semi-structured document-centric XML
- Data-centric is easily related to a data structure

```
class PO
{
  int id;
  Date submitted;
  Address billTo;
  Address shipTo;
  item order[];
}
```

Document- vs. Data-Centric XML

- Typically, documents for human (document-centric) consumption live a long time
- Some Data-Centric XML could live for a very short time
- Web services are *about Data-Centric* usage of XML

XML instances

- XML document contains an optional prolog
  - Identifies the document as an XML document
  - Includes any comments about the document
  - Includes any meta-information about the content of the document
- followed by a root element containing the document

XML Document prolog

Processing Instructions (PIs) are special directives to application that will process XML document:

```
<?PITarget . . . ?>
```

The only necessary PI – the XML declaration with the version of XML and the character encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by John Johnsson -->
   <po id="43871" submitted="2001-10-05">
         …
   </po>
```

Elements

- Element name may have all standard programming language identifier characters
- Elements can have 3 content types:
  - *Element-only* – consists entirely of nested elements
  - *Mixed content* – any combination of nested elements and text
  - *Empty content* – allows shorthand `<emptyElement/>` or

    `<message date="2004-0610"/>`

Elements

- XML elements are strictly nested and cannot overlap

```
<!-- Bad syntax -->
<P><I><B> Some text</I></B></P>
```

- There can be only one element at the very top level

```
<!-- This is not correct -->
<first>The first root element</first>
<second>The second root element</second>
```

Attributes

- The start tag of XML document may have attributes

```
<po id="4472" submitted="2004-06-05"> … </po>
```

- Attribute names starting with **`xml:`** are reserved
- Without any meta-information the attributes are considered just as text
- XML applications can attach any semantic information to markup

Attributes

```
<po id="43871" submitted="2001-10-05">
  <billTo>
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, </street>
    <street>Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
  </billTo>
  <shipTo>
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, </street>
    <street>Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
  </shipTo>
   ...
</po>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Attributes

```
<po id="43871" submitted="2001-10-05">
  <billTo id="addr-1" >
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, </street>
    <street>Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
  </billTo>
  <shipTo href="addr-1" />
  ...
</po>
```

Attributes vs. Elements

- One common rule is – to represent structured information using markups– easier to perform search
- However, decision depends on context and practical assumptions
- Generally whenever human creates a document attributes are mainly used, while computer generated documents mainly uses elements

Attributes vs. Elements

- Some of the problems with attributes are:
  - attributes cannot contain multiple values (child elements can)
  - attributes are not easily expandable (for future changes)
  - attributes cannot describe structures (child elements can)
  - attributes are more difficult to manipulate by program code
- There is no good way to answer this question

Improved Purchase Order

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by John Johnsson -->
<po id="44581" submitted="2004-06-05">
 <billTo id="addr-1">
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775>
 </billTo>
 <shipTo href="addr-1"/>
 <order>
    <item sku="316-BP" quantity="5">
      <description>Skateboard backpack</description>
    </item>
    <item sku="947-TI" quantity="5">
      <description> Street-style titanium skateboard.
      </description>
    </item>
 </order>
</po>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Character Data

- Any character outside the set of characters should be escaped and identified as a character reference: € or predefined escape sequence <
- XML text included into XML document as a text should be escaped:

```
<example>
 &lt;?xml version=&quot;1.0&quot;?&gt;
 &lt;root/&gt;
</example>
```

- CDATA section can be used

```
<example><![CDATA[
 <?xml version="1.0">
 <root/>
]]></example>
```

XML Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<message from="mm@imit.kth.se" to="order@statestown.com" sent="2004-06-05">
 <text>This is my order, MM</text>
 <attachments>
   <description>PO</description>
   <item>
    <po id="44581" submitted="2004-06-05">
     <billTo id="addr-1">
        <company>The Skateboard Warehouse</company>
        <street>One Warehouse Park, Building 17</street>
        <city>Boston</city>
        <state>MA</state>
        <postalCode>01775>
     </billTo>
     <shipTo href="addr-1"/>
     <order>
        <item sku="316-BP" quantity="5">
          <description>Skateboard backpack</description>
        </item>
        <item sku="947-TI" quantity="5">
          <description> Street-style titanium skateboard.</description>
        </item>
     </order>
    </po>
   </item>
 </attachments>
</message>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

XML Namespace

- Qualified name (QName) =

  Namespace identifier + Local name
- For identifiers, XML Namespaces uses Uniform Resource Identifiers (URI) :
  - Unified Resource Locator (URL) - **`http://www.imit.kth.se`**
  - Uniform Resource Names (URN) - **`urn:uuid:2FAC1234-31F8-11B4-A222-090909889098`**
- A Namespace identifier is associated with a prefix, which contains only XML element name characters (but not :)
- QName constructed as a combination of the prefix, “:” and the local name

```
newPrefix:newElementName
```

XML Namespaces

```
<msg:message from="mm@imit.kth.se" to="order@statestown.com" sent="2004-06-05"
    xmlns:msg="http://www.xcommerc.com/message"
    xmlns:po="http://statestown.com/ns/po" >
<msg:text>This is my order, MM</msg:text>
 <msg:attachments>
  <msg:description>PO</msg:description>
  <msg:item>
   <po:po id="44581" submitted="2004-06-05">
        <po:billTo id="addr-1">
                <po:company>The Skateboard Warehouse</po:company>
                <po:street>One Warehouse Park, Building 17</po:street>
                <po:city>Boston</po:city>
                <po:state>MA</po:state>
                <po:postalCode>01775</po:postalCode>
        </po:billTo>
        <po:shipTo href="addr-1"/>
        <po:order>
                <po:item sku="316-BP" quantity="5">
                  <po:description>Skateboard backpack</po:description>
                </po:item>
                <po:item sku="947-TI" quantity="5">
                  <po:description> Street-style titanium skateboard.
                  </po:description>
                </po:item>
        </po:order>
   </po:po>
  </msg:item>
 </msg:attachments>
</msg:message>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Default Namespaces

```
<message from="mm@imit.kth.se"
   to="order@statestown.com" sent="2004-06-05"
   xmlns="http://www.xcommerc.com/message"
   xmlns:po="http://statestown.com/ns/po" >
 <text>This is my order, MM</text>
 <attachments>
  <description>PO</description>
   <item>
     <po:po id="44581" submitted="2004-06-05">
       <po:billTo id="addr-1">
              . . .
       </po:billTo>
       <po:shipTo href="addr-1"/>
       <po:order>
              . . .
       </po:order>
     </po:po>
   </item>
 </attachments>
</message>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Nested Namespaces

```
<message from="mm@imit.kth.se" to="order@statestown.com"
   sent="2004-06-05"
   xmlns="http://www.xcommerc.com/message" >
<text>This is my order, MM</text>
 <attachments>
  <description>PO</description>
   <item>
   <po:po id="44581" submitted="2004-06-05"
            xmlns:po="http://statestown.com/ns/po" >
        <billTo id="addr-1">
                . . .
        </billTo>
        <shipTo href="addr-1"/>
        <order>
                . . .
        </order>
   </po:po>
   </item>
 </attachments>
</message>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Namespacing attributes

```
<message from="mm@imit.kth.se" to="order@statestown.com"
   sent="2004-06-05" xmlns="http://www.xcommerc.com/message">
<text>This is my order, MM</text>
 <attachments>
  <description>PO</description>
   <item>
    <po:po id="44581" submitted="2004-06-05"
           xmlns:po="http://statestown.com/ns/po"
           xmlns:p="http://statestown.com/ns/priority" >
        . . .
       <po:order>
          <po:item sku="316-BP" quantity="5" p:priority="high">
              <po:description>Skateboard backpack</po:description>
          </po:item>
          <po:item sku="947-TI" quantity="5" p:priority="low" >
               <po:description> Street-style titanium skateboard.
               </po:description>
          </po:item>
       </po:order>
     </po:po>
   </item>
 </attachments>
</message>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Well-Formedness

- If a document is conformant with the rules of XML syntax then it is said to be well-formed.
- If document is not well-formed then the XML parser reports error
- Parsers can support
  - Document Object Model (DOM) – build a tree structure in memory containing the XML document data
  - Simple API for XML (SAX) – generates events to applications when XML components (tags, text etc.) are encountered (application can listen for events)

Validity

- Validity is conformance with the applied format of the document:
  - what elements are allowed in the document
  - what is an order and relation between elements
  - what are attributes of every element
- Data Type Definitions (DTD) –describe an XML document structure (in most cases are not flexible enough – cannot be manipulated as XML documents, because they do not use XML syntax)
- XML Schema is alternative to DTD and expected to replace DTDs as primary means for describing XML document structure

XML Schema

- XML documents can refer to optional documents (Schemas) that specify how the XML document should be structured
- When the Schema document is provided validating parsers can check the XML document's structure against it
- Schemas are expressed in XML
- XML Schema vocabulary is itself defined using Schema
- The tags in the Schema have prefix **`xsd:`** (XML Schema Definition) which means that they are parts of the Schema (it is associated with **`http://www.x3.org/2001/XMLSchema`**)

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.x3.org/2001/XMLSchema"
   xmlns="http://www.skatestown.com/ns/po"
   targetNamespace ="http://www.skatestown.com/ns/po" >

   <xsd:annotation>
       <xsd:documentation xml:lang="en">
             This XML Schema document represents a PO
       </xsd:documentation>
   </xsd:annotation>
   …
</xsd:schema>
```

XML Schema and XML Documents

```
<?xml version="1.0" encoding="UTF-8"?>
<po:po xmlns:po="http://www.skatestown.com/ns/po"
  xmlns:xsi="http://www.x3.org/2001/XMLSchema-instance"
  xsi:schemaLocation= "http://www.skatestown.com/ns/po
                http://www.skatestown.com/schema/po.xsd"
      submitted="2004-06-10" id="12345">
            …
</po:po>
```

XML Schema Data Types

- Predefined types:
  - **`string, base64Binary, hexBinary, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, decimal, boolean, time, dateTime, duration, date, Name, QName, anyURI, ID, IDREF`**
- Constructed types:
  - They are constructed from predefined or already defined simple types
  - Types are restricted along a number of facets

Simple Types Usage

Definition:

```
<xsd:simpleType name="someCode">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="\[B-K]{2}\d{4}"/>
   </xsd:restriction>
 </xsd:simpleType>
```

Usage:

```
    . . .
<xsd:element name="courseName" type="xsd:string"/>
<xsd:element name="courseCode" type="someCode"/>
    . . .
```

Complex Types

- Type constructors groups:
  - **xsd:sequence**
  - **xsd:choice**
  - **xsd:all**
  - **xsd:group**
- Address Type:

```
<xsd:complexType name="addressType">
 <xsd:sequence>
   <xsd:element name="name" type="xsd:string" minOccurs="0"/>
   <xsd:element name="company" type="xsd:string" minOccurs="0"/>
   <xsd:element name="street" type="xsd:string" maxOccurs="unbound"/>
   <xsd:element name="city" type="xsd:string"/>
   <xsd:element name="state" type="xsd:string" minOccurs="0"/>
   <xsd:element name="postalCode" type="xsd:string" minOccurs="0"/>
   <xsd:element name="country" type="xsd:string" minOccurs="0"/>
 </xsd:sequence>
 <xsd:attribute name="id" type="xsd:ID"/>
 <xsd:attribute name="href" type="xsd:IDREF"/>
</xsd:complexType>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Complex Types

```
<xsd:complexType name="poType">
   <xsd:sequence>
        <xsd:element name="billTo" type="addressType"/>
        <xsd:element name="shipTo" type="addressType"/>
        <xsd:element name="order" >
             <xsd:complexType>
                  <xsd:sequence>
                       <xsd:element name="item"
                            type="itemType" maxOccurs="unbound"/>
                  </xsd:sequence>
             </xsd:complexType>
        </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="id" use="required"
                            type="xsd:positiveInteger"/>
   <xsd:attribute name="sumbitted" use="required"
                            type="xsd:date"/>
</xsd:complexType>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Putting things together

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.skatestown.com/ns/po">
   <xsd:annotation>
            <xsd:documentation xml:lang="en">Purchase order schema for SkatesTown.</xsd:documentation>
   </xsd:annotation>
   <xsd:element name="po" type="poType"/>
   <xsd:complexType name="poType">
            <xsd:sequence>
                  <xsd:element name"billTo" type"addressType"/>
                  <xsd:element name"shipTo" type"addressType"/>
                  <xsd:element name"order"/>
                        <xsd:complexType>
                              <xsd:sequence>
                                    <xsd:element name="item" type="itemType" maxOccurs="unbound"/>
                              </xsd:sequence>
                        </xsd:complexType>
                  </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="id" use="required",type="xsd:positiveInteger"/>
            <xsd:attribute name="submitted" use="required",type="xsd:date"/>
   </xsd:complexType>
   <xsd:complexType name="addressType">
            <xsd:sequence>
                  <xsd:element name="name" type="xsd:string" minOccurs="0"/>
                  <xsd:element name="company" type="xsd:string" minOccurs="0"/>
                  <xsd:element name="street" type="xsd:string" minOccurs="unbound"/>
                  <xsd:element name="city" type="xsd:string"/>
                  <xsd:element name="state" type="xsd:string" minOccurs="0"/>
                  <xsd:element name="postalCode" type="xsd:string" minOccurs="0"/>
                  <xsd:element name="country" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:ID"/>
            <xsd:attribute name="href" type="xsd:IDREF"/>
   </xsd:complexType>
   <xsd:complexType name="itemType">
             <xsd:sequence>
                   <xsd:element name="description" type="xsd:string" minOccurs="0"/>
             </xsd:sequence>
             <xsd:attribute name="sku" use="required"/>
                          <xsd:simpleType>
                                       <xsd:restriction base="xsd:string">
                                                    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
                                       </xsd:restriction>
                          </xsd:simpleType>
            <xsd:attribute name="quantity" use="required" type="xsd:positiveInteger"/>
   </xsd:complexType>
</xsd:schema>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Reuse

- Basic mechanism is **`ref`** attribute (reusability within the same document)
  - elements
  - model groups
  - attribute groups
- `<xsd:element ref="comment">` refers to
  `<xsd:element name="comment" type="xsd:string">`
- `<xsd:group ref="comment">` refers to
  `<xsd:group name="comment" > type </xsd:group>`
- `<xsd:attributeGroup name="referenceable" >` refers to
  `<xsd:attributeGroup name="referenceable" > type </xsd:attributeGroup>`

Reuse

- Reusability across documents
  - **`include`** – retrieving the definitions

    `<xsd:include schemaLocation="…"/>`
  - **`import`** – merges schemas from multiple namespaces into one schema

    `<xsd:import namespace="http://..."/>`

Include

Schema for Address

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.skatestown.com/ns/po">

        ...
 <complexType name="addressType">
   <xsd:sequence>
        <xsd:element name="name" type ="xsd:string" minOccurs="0"/>
    . . .
    </xsd:sequence>
 </xsd:element>
```

Schema for mailing list

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.skatestown.com/ns/mailingList">
 <xsd:include schemaLocation="http://www.skatestown.com/schema/address.xsd" />
        ...
 <xsd:element name="mailingList">
   <xsd:sequence>
        <xsd:element name="contact" type ="addressType" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xsd:sequence>
 </xsd:element>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Include

Usage:

```
<?xml version="1.0" encoding="UTF-8"?>
<list:mailingList xmlns:list="http://www.skatestown.com/ns/mailingList"
   xmlns:addr="http://www.skatestown.com/ns/po"
   xmlns:xsi="http://www.x3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.skatestown.com/ns/mailingList
                       http://www.skatestown.com/schema/mailingList.xsd
                       http://www.skatestown.com/ns/po
                       http://www.skatestown.com/schema/address.xsd" >
 <contact>
   <addr:company>The Skateboard Warehouse</addr:company>
   <addr:street>One Warehouse Park, Building 17</addr:street>
   <addr:city>Boston</addr:city>
   <addr:state>MA</addr:state>
   <addr:postalCode>01775</addr:postalCode>
 </contact>
</list:mailingList>
```

Import

Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:addr="http://www.skatestown.com/ns/po"
    xmlns:xsi="http://www.x3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.skatestown.com/ns/po
                  http://www.skatestown.com/schema/address.xsd"
    targetNamespace="http://www.skatestown.com/ns/mailingList">
    <xsd:import namespace="http://www.skatestown.com/ns/po" /> . . .
    <xsd:element name="mailingList">
       <xsd:sequence>
          <xsd:element name="contact" type ="addr:addressType"
                              minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:element>
```

Usage:

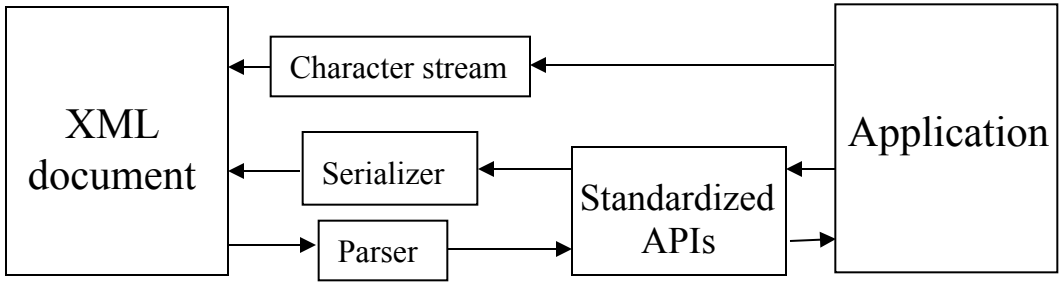
```
<?xml version="1.0" encoding="UTF-8"?>
<list:mailingList xmlns:list="http://www.skatestown.com/ns/mailingList"
    xmlns:xsi="http://www.x3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.skatestown.com/ns/mailingList
                        http://www.skatestown.com/schema/mailingList.xsd">
  <contact>
        <company>The Skateboard Warehouse</company>
        <street>One Warehouse Park, Building 17</street>
        <city>Boston</city>
        <state>MA</state>
        <postalCode>01775</postalCode>
  </contact>
</list:mailingList>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Extensions and Restrictions

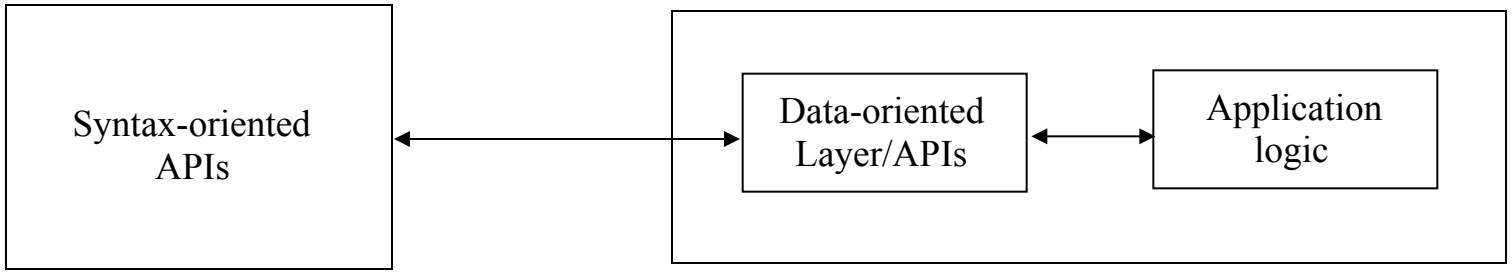
```
<xsd:simpleType name="StateType">
  <xsd:restriction base="xsd:string">
     <xsd:length value="2"/>
     <xsd:enumeration value="AR"/>
     <xsd:enumeration value="CA"/>
     <xsd:enumeration value="IL"/>
     <xsd:enumeration value="TX"/>
     <xsd:enumeration value="OK"/>
   </xsd:restriction>
</xsd:simpleType>
```

Processing XML



- Two basic approaches:
  - Document Object Model (DOM)
  - Simple API for XML (SAX)
- Parsing models:
  - Push
  - Pull
  - One-step
  - Hybrid

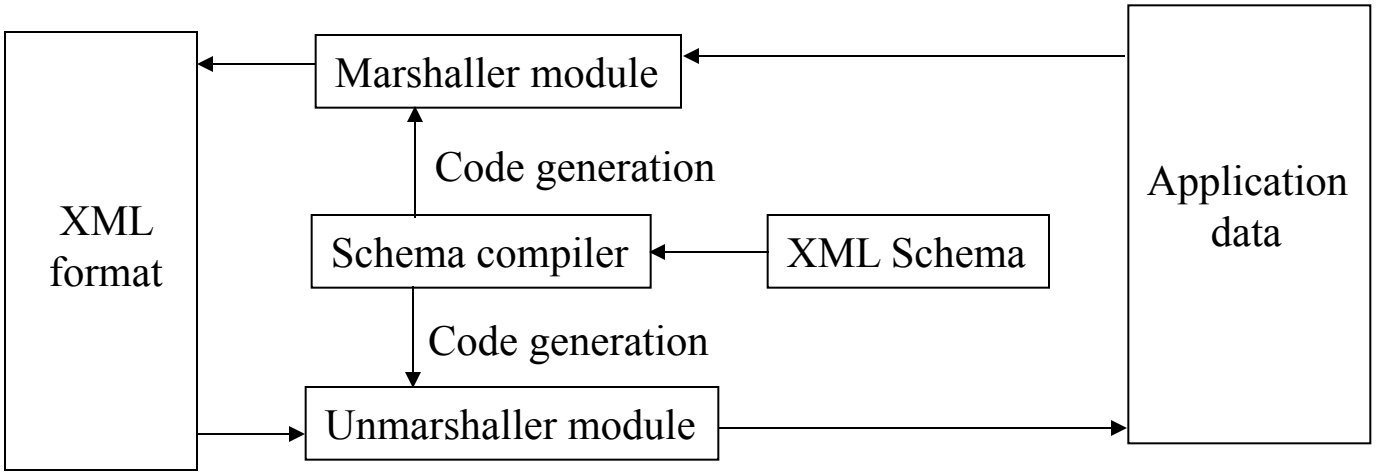
Processing XML



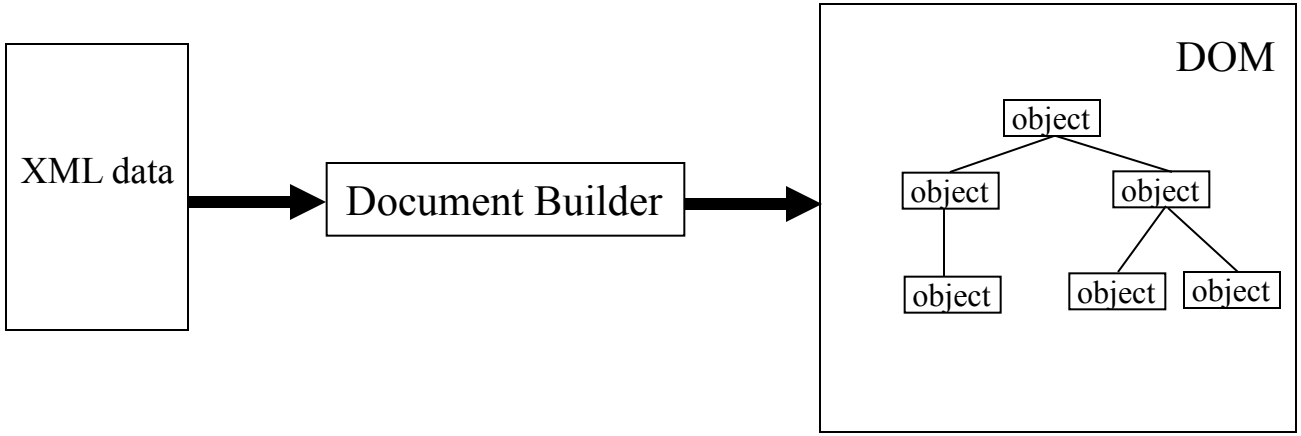
- Operation-centric
- Data-centric

Processing XML

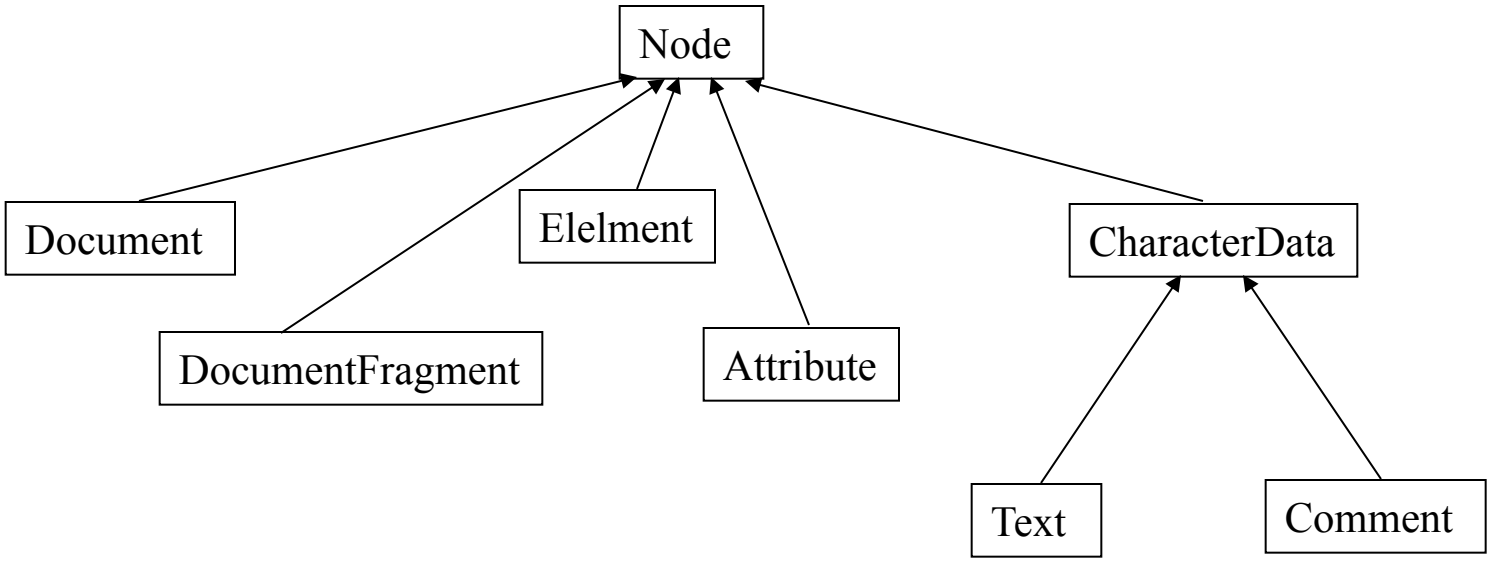
- Schema compiler



DOM parsing


XML data
Document Builder
DOM
object
object
object
object
object
object

DOM interfaces



DOM parsing

```
package com.madhu.xml;
import java.io.*;
import . . .
public class SimpleWalker {
    protected DocumentBuilder docBuilder;
    protected Element root;
    public SimpleWalker() throws Exception {
          DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
          docBuilder = dbf.newDocumentBuilder();
          DOMImplementation domImp = docBuilder.getDOMImplementation();
          if (domImp.hasFeature("XML", "1.0")){System.out.println("Message . . .");
          }
    }
    public void parse(String fileName) throws Exception {
          Document doc = docBuilder.parse(new FileInputStream(fileName));
          root = doc.getDocumentElement();
          System.out.println("Root element is " + root.getNodeName());
    }
    public void printAllElements() throws Exception {
          printElement("", root);
    }
    public void printElement(String indent, Node aNode) {
          if (aNode.getNodeType() == Node.TEXT_NODE) {
                System.out.println(indent + aNode.getNodeValue());
          } else {
                System.out.println(indent + "<" + aNode.getNodeName() + ">");
                Node child = aNode.getFirstChild();
                while (child != null) {printElement(indent + "\t", child);
                                child = child.getNextSibling();
                }
                System.out.println(indent + "</" + aNode.getNodeName() + ">");
          }
    }
    public static void main(String args[]) throws Exception {
          SimpleWalker sw = new SimpleWalker();
          sw.parse(args[0]);
          sw.printAllElements();
    }
}
```

From Ron Schmelzer et al. XML and Web Services Unleashed. Sams Publishing. 2002

DOM parsing

Input:

```
<library>
   <fiction>
        <book>Moby Dick</book>
        <book>The Last Trail</book>
   </fiction>
   <biography>
   <book>The Last Lion, Winston Spencer Churchill</book>
   </biography>
</library>
```

Output:

```
  <library>
     <fiction>
          <book>
                   Moby Dick
          </book>

          <book>
                   The Last Trial
          </book>

     </fiction>
     <biography>
          <book>
                   The Last Lion, Winston Spencer Churchill
          </book>
     </biography>
  </library>
```

Improved Purchase Order

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by John Johnsson -->
<po id="44581" submitted="2004-06-05">
 <billTo id="addr-1">
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775>
 </billTo>
 <shipTo href="addr-1"/>
 <order>
    <item sku="316-BP" quantity="5" unitPrice="49.95">
      <description>Skateboard backpack</description>
    </item>
    <item sku="947-TI" quantity="5" unitPrice="129.00">
      <description> Street-style titanium skateboard.
      </description>
    </item>
 </order>
 <tax>89.89</tax>
 <shippingAndHandling>200</shippingAndHandling>
 <totalCost>2087.64</totalCost>
</po>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Invoice CheckerDOM

```
package com.skatestown.invoice;
import …
//Check SkatesTown invoice totals using a DOM parser.
public class InvoiceCheckerDOM implements InvoiceChecker {
    /** Check invoice totals.*/
    public void checkInvoice(InputStream invoiceXML)  throws Exception {
        // Invoice running total
        double runningTotal = 0.0;
        // Obtain parser instance and parse the document
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(invoiceXML);
        // Calculate order subtotal
        NodeList itemList = doc.getElementsByTagName("item");
        for (int i = 0; i < itemList.getLength(); i++) {
            // Extract quantity and price
            Element item = (Element)itemList.item(i);
            Integer qty = Integer.valueOf(item.getAttribute("quantity"));
            Double price = Double.valueOf(item.getAttribute("unitPrice"));
            // Add subtotal to running total
            runningTotal += qty.intValue() * price.doubleValue();
        }
        // Add tax
        Node nodeTax = doc.getElementsByTagName("tax").item(0);
        runningTotal += doubleValue(nodeTax);
        // Add shipping and handling
        Node nodeShippingAndHandling =doc.getElementsByTagName("shippingAndHandling").item(0);
        runningTotal += doubleValue(nodeShippingAndHandling);
        // Get invoice total
        Node nodeTotalCost = doc.getElementsByTagName("totalCost").item(0);
        double total = doubleValue(nodeTotalCost);
        if (Math.abs(runningTotal - total) >= 0.005) {
            throw new Exception("Invoice error: total is " + Double.toString(total) + " while our
            calculation shows a total of " +Double.toString(Math.round(runningTotal * 100) / 100.0));
        }
    }
private double doubleValue(Node node) throws Exception {
        // Get the character data from the node and parse it
        String value = ((CharacterData)node.getFirstChild()).getData();
        return Double.valueOf(value).doubleValue();
    }
}
```

Creating XML Document

```
package com.madhu.xml;
import java.io.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
public class DocBuilder {
    protected DocumentBuilder docBuilder;
    protected Element root;
    protected Document doc;
    protected PrintWriter writer;
    public DocBuilder() throws Exception {
          DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
          docBuilder = dbf.newDocumentBuilder();
    }
    public void buildDOM(String startDir) throws Exception {
          doc = docBuilder.newDocument();
          root = doc.createElement("directory-listing");
          appendFile(root, new File(startDir));
          doc.appendChild(root);
    }
    public void appendFile(Node parent, File aFile) throws Exception {
          if (aFile.isDirectory()) {
                    Element dirElement = doc.createElement("directory");
                    dirElement.setAttribute("name", aFile.getName());
                    File[] files = aFile.listFiles();
                    int n = files.length;
                    for (int i=0; i<n; i+=1) {appendFile(dirElement, files[i]);}
                    parent.appendChild(dirElement);
          } else {
                    Element fileElement = doc.createElement("file");
                    Text fileName = doc.createTextNode(aFile.getName());
                    fileElement.appendChild(fileName);
                    parent.appendChild(fileElement);
          }
    }
```

From Ron Schmelzer et al. XML and Web Services Unleashed. Sams Publishing. 2002

Creating XML Document

```
public void writeDOM(PrintWriter bw) throws Exception {
        writer = bw;
        writer.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
        writeNode("", root);
}
public void writeNode(String indent, Node aNode) {
        switch (aNode.getNodeType()) {
                case Node.TEXT_NODE:
                        writer.println(indent + aNode.getNodeValue());
                        break;
                case Node.ELEMENT_NODE:
                        writer.print(indent + "<" + aNode.getNodeName());
                        NamedNodeMap attrs = aNode.getAttributes();
                        int n = attrs.getLength();
                        for (int i=0; i<n; i+=1) {
                                Node attr = attrs.item(i);
                                writer.print(" " + attr.getNodeName() + "=\"");
                                writer.print(attr.getNodeValue() + "\"");
                        }
                        writer.println(">");
                        Node child = aNode.getFirstChild();
                        while (child != null) {
                                writeNode(indent + "\t", child);
                                child = child.getNextSibling();
                        }
                        writer.println(indent + "</" + aNode.getNodeName() + ">");
                        break;
        }
}
public static void main(String args[]) throws Exception {
        DocBuilder db = new DocBuilder();
        db.buildDOM(args[0]);
        PrintWriter bw = new PrintWriter(new FileWriter(args[1]));
        db.writeDOM(bw);
        bw.close();
}
}
```

From Ron Schmelzer et al. XML and Web Services Unleashed. Sams Publishing. 2002

DocBuilder Output

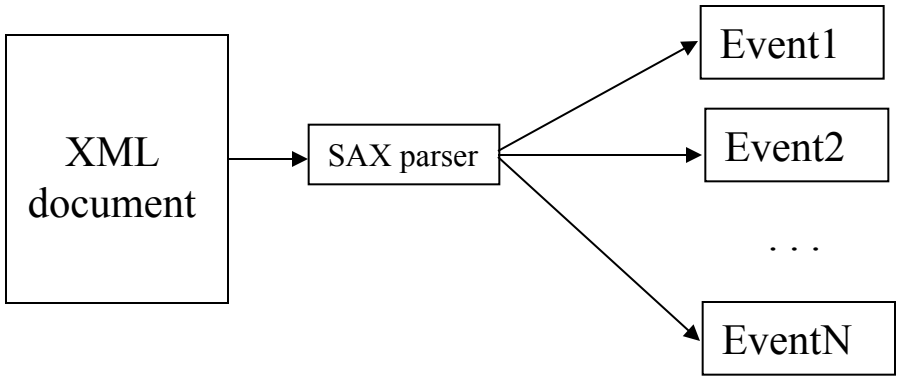
```
<directory-listing>
    <directory name="..">
          <directory name="com">
                    <directory name="madhu">
                              <directory name="xml">
                                        <file>
                                                  DocBuilder.class
                                        </file>
                                        <file>
                                                  SimpleWalker.class
                                        </file>
                              </directory>
                    </directory>
          </directory>
          <directory name="test">
                    <file>
                              Makefile
                    </file>
                    <file>
                              personal-schema.xml
                    </file>
                    <file>
                              personal.dtd
                    </file>
                    <file>
                              personal.xml
                    </file>
                    <file>
                              personal.xsd
                    </file>
          </directory>
           .  .  .
```

DOM drawbacks

- DOM can be memory intensive
- DOM API is a bit complex and is not practical for small devices (PDAs, phones …)

SAX

- SAX is an API that can be used to parse XML document
- It provides a framework for defining event listeners or handlers



SAX (example of events)

Input:

```
<?xml version="1.0" encoding="UTF-8"?>
  <fiction>
      <book>Moby Dick</book>
  </fiction>
```

SAX events:

```
Start document
Start element: fiction
Start element: book (including attributes)
Characters: Moby Dick
End element: book
End element: fiction
End document
```

Working with SAX

```
package …
import …
public class SAXDemo extends DefaultHandler {
    public void startDocument() {
          System.out.println("***Start of Document***");
    }
    public void endDocument() {
          System.out.println("***End of Document***");
    }
    public void startElement(String uri, String localName, String qName, Attributes attributes) {
          System.out.print("<" + qName);
          int n = attributes.getLength();
          for (int i=0; i<n; i+=1) {
               System.out.print(" "+attributes.getName(i)+"='"+attributes.getValue(i) + "'");
          }
          System.out.println(">");
    }
    public void characters(char[] ch, int start, int length) {
          System.out.println(new String(ch, start, length).trim());
    }
    public void endElement(String namespaceURI, String localName, String qName) throws
                    SAXException {
          System.out.println("</" + qName + ">");
    }
    public static void main(String args[]) throws Exception {
          if (args.length != 1) {
               System.err.println("Usage: java SAXDemo <xml-file>");
               System.exit(1);
          }
          SAXDemo handler = new SAXDemo();
          SAXParserFactory factory = SAXParserFactory.newInstance();
          SAXParser parser = factory.newSAXParser();
          parser.parse(new File(args[0]), handler);
    }
}
```

From Ron Schmelzer et al. XML and Web Services Unleashed. Sams Publishing. 2002

Improved Purchase Order

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by John Johnsson -->
<po id="44581" submitted="2004-06-05">
 <billTo id="addr-1">
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775>
 </billTo>
 <shipTo href="addr-1"/>
 <order>
    <item sku="316-BP" quantity="5" unitPrice="49.95">
      <description>Skateboard backpack</description>
    </item>
    <item sku="947-TI" quantity="5" unitPrice="129.00">
      <description> Street-style titanium skateboard.
      </description>
    </item>
 </order>
 <tax>89.89</tax>
 <shippingAndHandling>200</shippingAndHandling>
 <totalCost>2087.64</totalCost>
</po>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Invoicing with SAX

```
package com.skatestown.invoice;
import java.io.InputStream;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.helpers.DefaultHandler;
// * Check SkatesTown invoice totals using a SAX parser.
public class InvoiceCheckerSAX extends DefaultHandler implements InvoiceChecker {
    // invoice running total
    double runningTotal = 0.0;
    // invoice total
    double total = 0.0;
    // Utility data for extracting money amounts from content
    boolean isMoneyContent = false;
    double amount = 0.0;
    // Check invoice totals.      * @param invoiceXML Invoice XML document  *
    // @exception Exception Any exception returned during checking
    public void checkInvoice(InputStream invoiceXML) throws Exception {
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();
        // Parse the input; we are the handler of SAX events
        saxParser.parse(invoiceXML, this);
    }
    // SAX DocumentHandler methods
    public void startDocument() throws SAXException {
        runningTotal = 0.0;
        total = 0.0;
        isMoneyContent = false;
    }
```

Invoicing with SAX cntd.

```
  public void endDocument() throws SAXException {
  // Use delta equality check to prevent cumulative binary arithmetic errors
    if (Math.abs(runningTotal - total) >= 0.005) {
            throw new SAXException( "Invoice error: total is " + Double.toString(total) + "
    while our calculation are " + Double.toString(Math.round(runningTotal * 100) / 100.0));
     }
   }
    public void startElement(String namespaceURI, String localName,
                    String qualifiedName, Attributes attrs) throws SAXException {
        if (localName.equals("item")) {
             runningTotal +=  Integer.valueOf(attrs.getValue(namespaceURI,
            "quantity")).intValue() * Double.valueOf(attrs.getValue(namespaceURI,
            "unitPrice")).doubleValue();
        } else if (localName.equals("tax") || localName.equals("shippingAndHandling") ||
          localName.equals("totalCost")) { // Prepare to extract money amount
            isMoneyContent = true;
        }
    }
    public void endElement(String namespaceURI,  String localName,  String qualifiedName)
                    throws SAXException {
        if (isMoneyContent) {
            if (localName.equals("totalCost")) {
                total = amount;
            } else {            // It must be tax or shippingAndHandling
                runningTotal += amount;
            }
            isMoneyContent = false;
        }
    }
    public void characters(char buf[], int offset, int len)
                    throws SAXException {
        if (isMoneyContent) {
            String value = new String(buf, offset, len);
            amount = Double.valueOf(value).doubleValue();
        }
    }
}
```

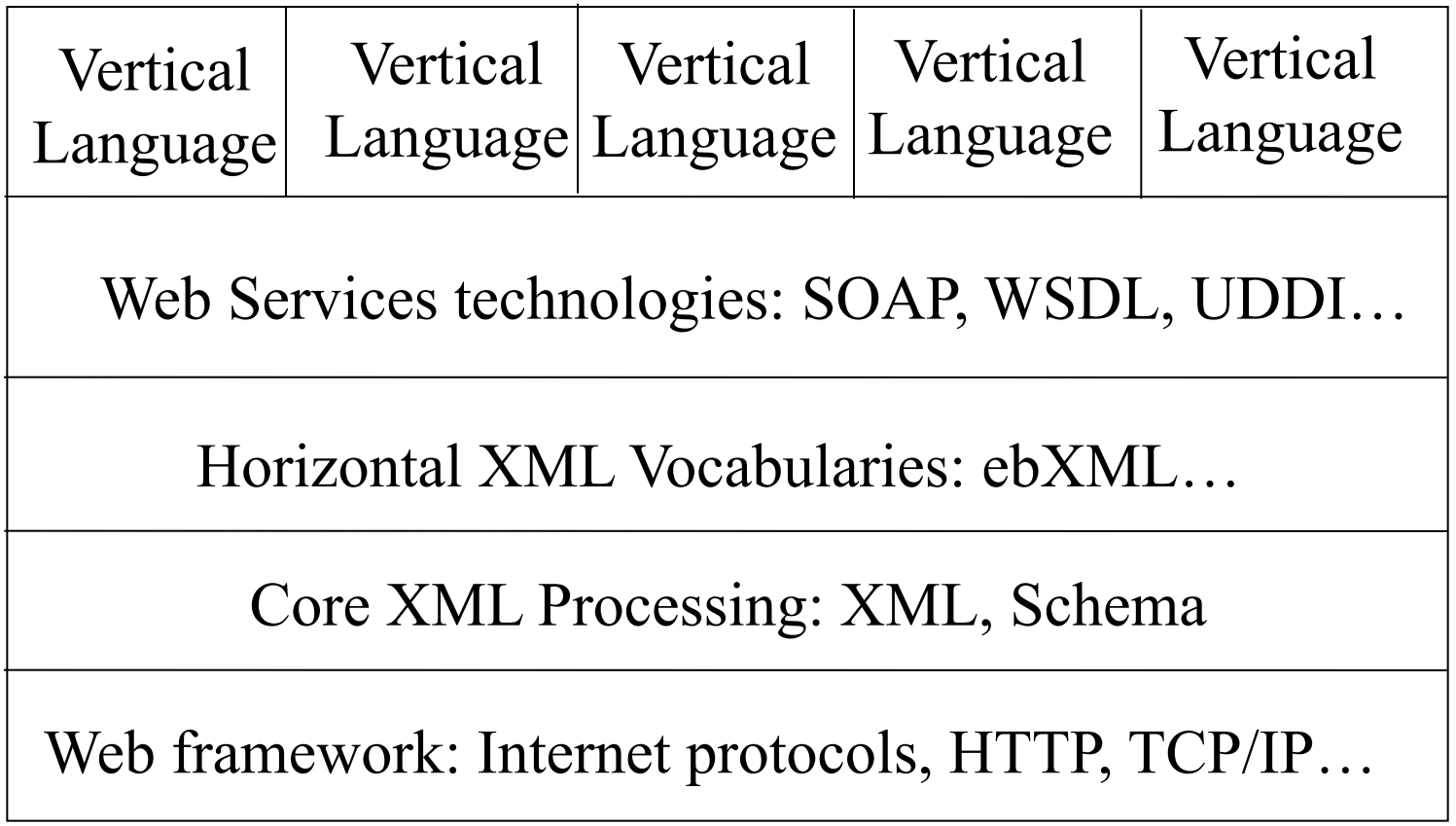
SAX vs. DOM

- DOM is in memory structure, SAX is not
- SAX is simpler than DOM in many ways
- SAX is event-based API
- DOM parses in space while SAX parses in time

SAX disadvantages

- It can be a bit harder to visualize
- SAX parsing is “single pass”
- No random access at all
- SAX parsers are mostly read-only – do not provide ability to manipulate document
- No formal specification for SAX

XML (Web Services protocol stack)



JSON (JavaScript Object Notation)

- JSON is lightweight text-data interchange format
- JSON syntax is a subset of the JavaScript object notation syntax.
- Data is in name/value pairs
- Data is separated by comma
- Curly brackets holds objects
- Square brackets holds arrays

```
{"employees": [
  { "firstName":"John" , "lastName":"Doe" },
  { "firstName":"Anna" , "lastName":"Smith" },
  { "firstName":"Peter" , "lastName":"Jones" }
  ]
}
```

Adopted from:http://www.w3schools.com/json/default.asp

JSON

- JSON Values:
  - A number (integer or floating point)
  - A string (in double quotes)
  - A Boolean (true or false)
  - An array (in square brackets)
  - An object (in curly brackets)
  - Null
- Objects can contain multiple name/values pairs:

```
{ "firstName":"John", "lastName":"Doe" }
```

This equals to the JavaScript statements:

```
firstName = "John"    lastName = "Doe"
```

JSON arrays are written inside square brackets :

```
{"employees": [{ "firstName":"John" , "lastName":"Doe" },
               { "firstName":"Anna" , "lastName":"Smith" },
               { "firstName":"Peter" , "lastName":"Jones" }]}
```

Adopted from:http://www.w3schools.com/json/default.asp

JSON vs. XML

JSON:

```
{"menu":
    {"id": "file",
    "value": "File",
    "popup": {
         "menuitem": [
          {"value": "New", "onclick": "CreateNewDoc()"},
          {"value": "Open", "onclick": "OpenDoc()"},
          {"value": "Close", "onclick": "CloseDoc()"}
        ]
        }
        }}
```

XML:

```
<menu id="file" value="File">
        <popup>
                <menuitem value="New" onclick="CreateNewDoc()" />
                <menuitem value="Open" onclick="OpenDoc()" />
                <menuitem value="Close" onclick="CloseDoc()" />
         </popup>
</menu>
```

Adopted from:http://www.w3schools.com/json/default.asp

JSON Schema

```
{"name": "Product",
  "properties": {
    "id": {"type": "number", "description": "Product identifier",
                        "required": true},
    "name": {"type":"string","description":"Name of the product",
                              "required": true},
     "price": {"type": "number", "minimum": 0, "required":true},
     "tags": {"type": "array", "items": {"type":"string"}},
     "stock": {"type": "object", "properties": {"warehouse":
        {"type": "number"}, "retail":{"type":"number"}}}}}
```

JSON vs. XML

- JSON looks more like the data structures in programming languages. It has less redundant repetition of names.
- Parsing in JSON is easy for JavaScript
- JSON is good for transferring data in textual format, however, it is not intended for presentation/specification
- JSON doesn't have namespaces and this creates problem for merging documents and for structuring documents
- It is possible represent the entire design of a system as XML
- It is not the question of XML vs. JSON; It is the question of When XML and When JSON?

Next Lecture

- SOAP

Text book **Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2nd Edition**

Chapter 3