Service-Oriented Architecture

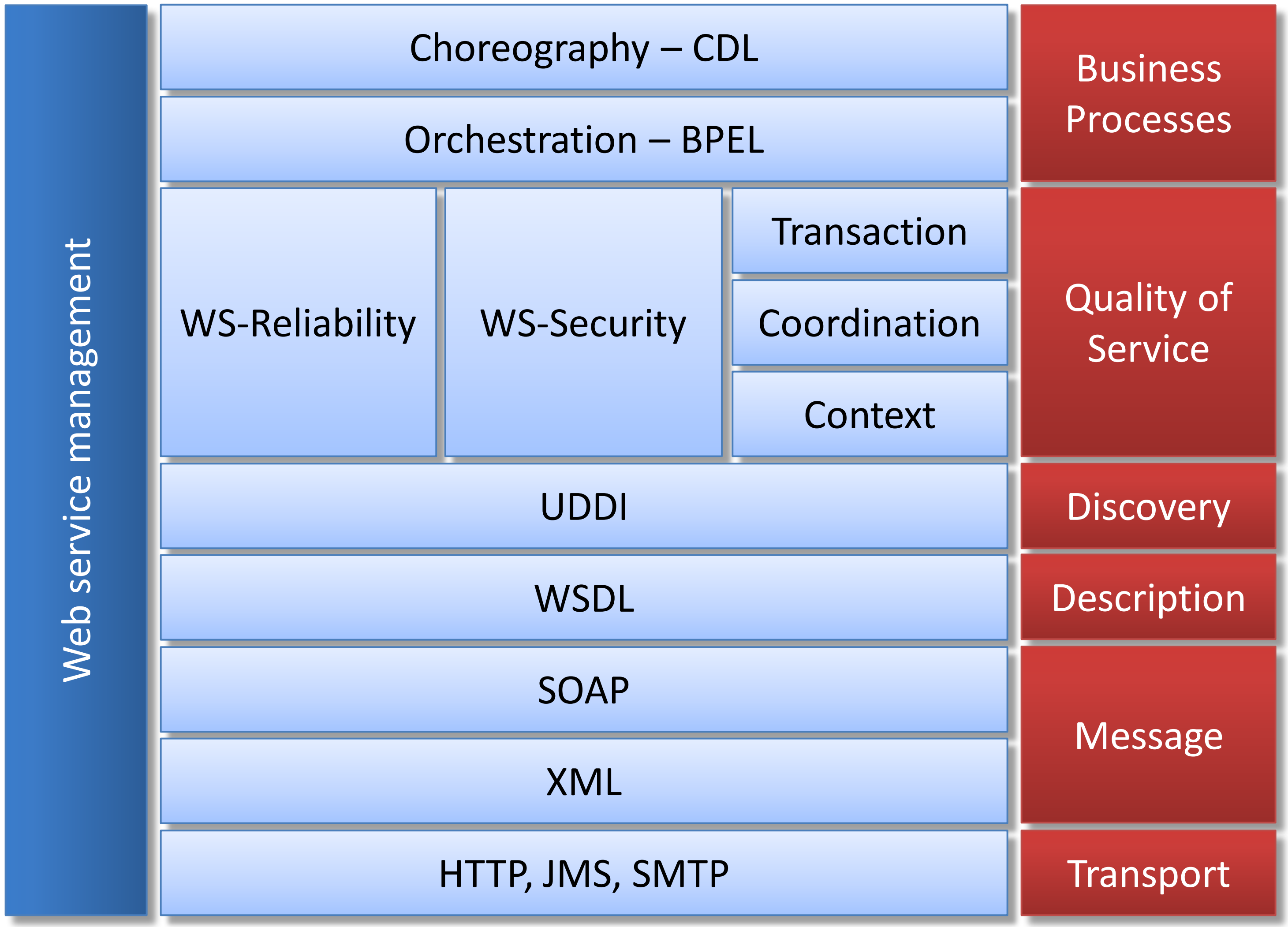
WS Technology Stack

Lecture 02

Henry Novianus Palit

hnpalit@petra.ac.id

Web Service Technology Stack *(1)*



Web Service Technology Stack *(2)*

- Enabling technology standards
  - At the transport level, Web services take advantage of HTTP
  - XML (eXtensible Markup Language) is a widely accepted format for all exchanging data and its corresponding semantics; XML is the fundamental building block for nearly every other layer in the Web services stack
- Core service standards
  - Communication protocol → SOAP (Simple Object Access Protocol) is a simple XML-based messaging protocol on which Web services rely to exchange information among themselves; it implements a request/response model and uses HTTP to penetrate firewalls
  - Service description → WSDL (Web Services Description Language) defines the XML grammar for describing services as collections of communicating endpoints capable of exchanging messages
  - Service publication → UDDI (Universal Description, Discovery, and Integration) is a public directory that provides publication of online services and facilitates eventual discovery of Web services

Web Service Technology Stack *(3)*

- Service composition and collaboration standards
  - Service composition → this family of standards describes the execution logic of Web service based applications by <u>defining their control flows</u> (such as conditional, sequential, parallel, and exceptional execution) and <u>prescribing the rules</u> for consistently managing their unobservable business data; BPEL (Business Process Execution Language) is the *de facto* standard
  - Service collaboration → this standard describes cross-enterprise collaborations of Web service participants by <u>defining their common observable behavior</u>, where synchronized information exchanges occur through their shared contact points, when commonly defined ordering rules are satisfied; WS-CDL (Web Services Choreography Description Language) supports service collaboration

Web Service Technology Stack *(4)*

- Service composition and collaboration standards (cont'd)
  - Coordination/transaction standards → WS-Coordination and WS-Transaction complement BPEL to <u>provide mechanisms for defining specific standard protocols</u> for use by transaction processing systems, workflow systems, or other applications that wish <u>to coordinate multiple Web services</u>
  - Value-added standards → additional elements <u>supporting complex business interactions</u> (e.g., WS-Security, WS-Policy and WS-Management) that Web services can use <u>to automate truly critical business processes</u>; the standards include mechanisms for security and authentication, authorization, trust, privacy, secure conversations, contract management, and so on

Service-Oriented Architecture

Overview of XML

Lecture 02

Henry Novianus Palit
hnpalit@petra.ac.id

XML Document Structure *(1)*

- XML is a language used for the description and delivery of marked up electronic text over the Web
- Two distinct characteristics of XML (compared to other markup languages):
  - The notion of a *document type*
    XML's constituent parts and their structure formally define the type of a document
  - The concept of *portability*
    All XML documents, whatever language or writing system they employ, use the same underlying character encoding scheme to ensure that documents are portable between different computing environments
- An XML document is composed of named containers and their contained data values; the containers are represented as *declarations*, *elements*, and *attributes*

XML Document Structure *(2)*

- An XML document is also known as an instance or *XML document instance*

```
<?xml version="1.0" encoding="UTF-8"?>
<BillingInformation>
    <Name> Auto Parts and Car Accessories </Name>
    <BillingDate> 2008-09-15 </BillingDate>
    <Address>
        <Street> 158 Edward St. </Street>
        <City> Brisbane </City>
        <State> QLD </State>
        <PostalCode> 4000 </PostalCode>
    </Address>
</BillingInformation>
```

XML Document Structure *(3)*

- XML declaration
  - It comprises a few lines at the beginning of an XML document
  - The XML processing software uses the declaration to <u>determine how to deal with the subsequent XML content</u>
  - A typical XML declaration begins with a *prologue* that contains a declaration of conformity to version 1.0 of the XML standard and to the UTF-8 encoding standard
- XML elements
  - An element is <u>a fundamental data container of an XML message</u>
  - It can be declared to <u>contain a (char) data value, another element, a combination of a data value & another element, or it may be empty</u>
  - The topmost element of the XML document is called the *root element*
  - Elements contained in other elements are referred to as *nested elements*; the containing is the parent and the nested is the child
  - Data values contained within a document are known as the *content* of the document; when the elements'/attributes' names are descriptive, the content becomes intuitive and self-explanatory (self-describing)

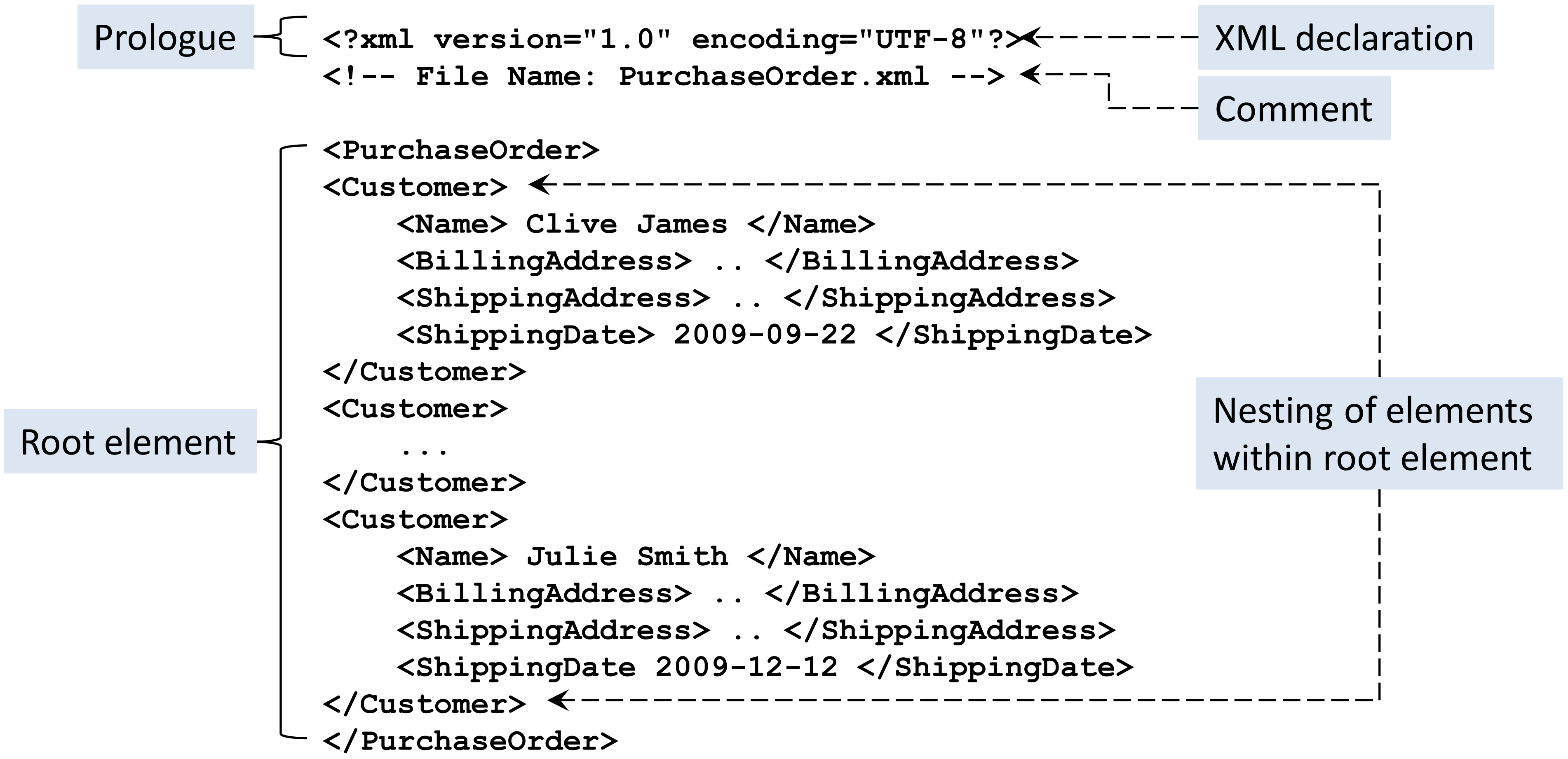
XML Document Structure *(4)*

- XML attributes
  - Another way of putting data into an XML document, attributes are used to better specify the content of an element on which they appear by adding information about a defined element
  - Each attribute is a name-value pair where the value must be in either single or double quotes
  - Unlike elements, attributes cannot be nested
  - Attributes must always be declared in the start tag of an element

```
<?xml version="1.0" encoding="UTF-8"?>
<BillingInformation customer-type="manufacturer">
    <Name> Auto Parts and Car Accessories </Name>
    <BillingDate> 2008-09-15 </BillingDate>
    <Address>
        <Street> 158 Edward St. </Street>
        <City> Brisbane </City>
        <State> QLD </State>
        <PostalCode> 4000 </PostalCode>
    </Address>
</BillingInformation>
```

XML Document Structure *(5)*

- Layout of typical XML document



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File Name: PurchaseOrder.xml -->

<PurchaseOrder>
<Customer>
    <Name> Clive James </Name>
    <BillingAddress> .. </BillingAddress>
    <ShippingAddress> .. </ShippingAddress>
    <ShippingDate> 2009-09-22 </ShippingDate>
</Customer>
<Customer>
    ...
</Customer>
<Customer>
    <Name> Julie Smith </Name>
    <BillingAddress> .. </BillingAddress>
    <ShippingAddress> .. </ShippingAddress>
    <ShippingDate 2009-12-12 </ShippingDate>
</Customer>
</PurchaseOrder>
```

XML Namespaces *(1)*

- As XML allows document designers to choose their own names, it is possible that name clashes occur when two designers choose the same tag names for their elements
- XML namespaces provide a way to distinguish elements that use the same local name but are in fact different, e.g., a namespace can identify whether an address is a postal address, an email address, or an IP address
- Namespaces in XML provide a facility for associating the elements and/or attributes in all or part of a document with a particular schema
- Tag names within a namespace must be unique
- All namespace declarations have a scope, covering the element on which it is declared and the element's children

XML Namespaces *(2)*

- The namespace name and the local name of the element together form a globally unique name known as a *qualified name*, often referred to as *QName* and consisting of a (namespace) prefix & the local name separated by a colon
- A namespace declaration in XML is <u>indicated by a Uniform Resource Identifier (URI), denoting the namespace name</u>
- The URIs are simply used for identification & scoping purposes and <u>it is not necessary that they point to any actual resources</u>
- <u>The URI may be mapped to a prefix</u> that may then be used in front of tag and attribute names, separated by a colon
  E.g., **`xmlns:<namespace prefix> = <some URI>`**
- When the prefix is attached to local names of elements and attributes, they are associated with the correct namespace

XML Namespaces *(3)*

- Example of an XML document using namespaces

```
<?xml version="1.0" encoding="UTF-8"?>
<bi:BillingInformation customer-type="manufacturer"
    xmlns:bi="http://www.auto-parts.com/BillingInfo"
    xmlns:addr="http://www.auto-parts.com/Address">

    <bi:Name> Auto Parts and Car Accessories </bi:Name>
    <addr:Address>
        <addr:Street> 158 Edward St. </addr:Street>
        <addr:City> Brisbane </addr:City>
        <addr:State> QLD </addr:State>
        <addr:PostalCode> 4000 </addr:PostalCode>
    </addr:Address>
    <bi:BillingDate> 2002-09-15 </bi:BillingDate>
</bi:BillingInformation>
```

XML Schema *(1)*

- A way to define XML tags and structure, schemas provide much needed capabilities for expressing XML documents and provide support for metadata characteristics such as structural relationship, cardinality, valid values, and data types
- The term *schema* when used in XML refers to a document that defines the content of, the structure of, the class of XML documents
- XSD (XML Schema Definition Language) provides a very powerful and flexible way to validate XML documents
- XSD includes facilities for declaring elements and attributes, reusing elements from other schemas, defining complex element definitions, and defining restrictions for even the simplest of data types

XML Schema *(2)*

- An XML schema is made up of *schema components*, which include the following
  - Data types that embrace both simple & complex/composite and extensible data types
  - Element type and attribute declarations
  - Constraints
  - Relationships that express association between elements
  - Namespaces and import/include options to support modularity, as they make it possible to include reusable structures, containers, and custom data types through externally managed XML schemas
- Since schemas can more clearly define the types of data that are to be contained in an XML document, they allow for a closer check on the accuracy of XML documents

Example of XML Schema *(1)*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:PO="http://www.auto-parts.com/PurchaseOrder"
  targetNamespace="http://www.auto-parts.com/PurchaseOrder">

  <!-- Purchase Order schema -->
  <xsd:element name="PurchaseOrder" type="PO:PurchaseOrderType"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:all>
      <xsd:element name="ShippingInformation" type="PO:Customer"
           minOccurs="1" maxOccurs="1"/>
      <xsd:element name="BillingInformation"  type="PO:Customer"
           minOccurs="1" maxOccurs="1"/>
      <xsd:element name="Order"               type="PO:OrderType"
           minOccurs="1" maxOccurs="1"/>
    </xsd:all>
  </xsd:complexType>
```

(to be continued)

Example of XML Schema *(2)*

```
<xsd:complexType name="Customer">
  <xsd:sequence>
    <xsd:element name="Name" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Address" type="PO:AddressType"
         minOccurs="1" maxOccurs="1"/>
    <xsd:choice>
      <xsd:element name="BillingDate"  type="xsd:date"/>
      <xsd:element name="ShippingDate" type="xsd:date"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AddressType">
  <xsd:sequence>
    <xsd:element name="Street"     type="xsd:string"/>
    <xsd:element name="City"       type="xsd:string"/>
```

(to be continued)

Example of XML Schema *(3)*

```
    <xsd:element name="State"      type="xsd:string"/>
    <xsd:element name="PostalCode" type="xsd:decimal"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="OrderType">
  <xsd:sequence>
    <xsd:element name="Product" type="PO:ProductType"
         minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="ItemsSold" type="xsd:positiveInteger"/>
</xsd:complexType>
```

(to be continued)

Example of XML Schema *(4)*

```
  <xsd:complexType name="ProductType">
    <xsd:attribute name="Name" type="xsd:string"/>
    <xsd:attribute name="Price">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Quantity" type="xsd:positiveInteger"/>
  </xsd:complexType>

</xsd:schema>
```

Example of XML Document Instance *(1)*

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

<PO:PurchaseOrder
  xmlns:PO="http://www.auto-parts.com/PurchaseOrder"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.auto-parts.com/purchaseOrder.xsd">

  <ShippingInformation>
    <Name> Auto Parts and Car Accessories </Name>
    <Address>
      <Street> 459 Wickham St. </Street>
      <City> Fortitude Valley </City>
      <State> QLD </State>
      <PostalCode> 4006 </PostalCode>
    </Address>
    <ShippingDate> 2010-09-20 </ShippingDate>
  </ShippingInformation>
```

(to be continued)

Example of XML Document Instance *(2)*

```
  <BillingInformation>
    <Name> Auto Parts and Car Accessories </Name>
    <Address>
      <Street> 158 Edward St. </Street>
      <City> Brisbane </City>
      <State> QLD </State>
      <PostalCode> 4000 </PostalCode>
    </Address>
    <BillingDate> 2010-09-15 </BillingDate>
  </BillingInformation>

  <Order Total="32000.00" ItemsSold="200">
    <Product Name="Catalytic Converter" Price="240.00"
      Quantity="100"/>
    <Product Name="Drive Axle"          Price="80.00"
      Quantity="100"/>
  </Order>
</PO:PurchaseOrder>
```

Type Definitions & Declarations *(1)*

- XSD introduces a sharp distinction between
  - Definitions → create new types (both simple and complex)
  - Declarations → enable elements and attributes with specific names and types (both simple and complex) to appear in document instances
- Schema type definitions in XSD are
  - Complex types → define their contents in terms of elements that may consist of further elements and attributes
  - Simple types → define their content in terms of elements and attributes that can contain only data
- Element declaration
  - Elements – the primary ingredients of an XML schema – can be declared using the **`<xsd:element>`** construct
  - It defines the element name, content model, and allowable attributes & data types for each element type
  - W3C XML schemas provide extensive data type support, including numerous built in and derived data types that can be applied as constraints to any element or attribute

Type Definitions & Declarations *(2)*

- Element declaration (cont'd)
  - Element declarations that appear as immediate descendants of the **`<xsd:schema>`** element are known as *global element declarations* and can be referenced from anywhere within the schema document or from other schemas, e.g., **`PurchaseOrderType`**
  - Elements declarations that appear as part of complex type definitions – either directly or indirectly – are called *local element declarations*, e.g., **`Customer`** and **`ProductType`**
  - A construct that declares an element content may use *compositors* to aggregate existing types into a structure; there are three types of compositors that can be used within XML schemas
    - **`sequence`** → the sequence of individual elements defined within a complex type or group must be followed by the XML document
    - **`choice`** → the document designer makes a choice between a number of defined options in a complex type or group
    - **`all`** → all the elements contained in a complex type or group may appear once or not at all, and may appear in any order

Type Definitions & Declarations *(3)*

- Attribute declaration
  - The **`<xsd:attribute>`** element is used to indicate that a complex element has an attribute
  - An attribute must have one of these simple types: **`boolean`**, **`byte`**, **`date`**, **`dateTime`**, **`decimal`**, **`double`**, **`duration`**, **`float`**, **`integer`**, **`language`**, **`long`**, **`short`**, **`string`**, **`time`**, **`token`**, etc.
- Simple types
  - XML Schema allows users to define their own custom simple types by creating a **`simpleType`** element with one of the supported data types as a base and adding constraining facets to it
- Complex types
  - The **`complexType`** element is used to define structured types, containing child elements and/or attributes
  - Complex type definitions appear as children of an **`<xsd:schema>`** element and can be referenced from elsewhere in the schema and from other schemas

Reuse of XML Schemas

- Benefits of reusing components of XML schemas include
  - Shorter development cycles
  - Reducing application development costs
  - Simpler maintenance of code
  - Promoting the use of enterprise data standards
- Techniques to reuse XML schema components
  - Deriving complex types $\rightarrow$ complex types are derived from other existing types (simple or complex) by extension or by restriction
  - Including and importing schemas $\rightarrow$ cross-domain reuse supported in W3C XML schemas allows an XML schema (or parts thereof) to be referenced and its content to be reused by other XML schemas; combining schemas in the XSD can be achieved by using the **`include`** and the **`import`** elements

Complex Type Extension *(1)*

- Adding attributes may extend complex types but one cannot modify or remove existing attributes
- The XML processor handles the extensions by appending the new content model after the base type's content model

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:PO="http://www.auto-parts.com/PurchaseOrder"
  targetNamespace="http://www.auto-parts.com/PurchaseOrder">

  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element name="Number" type="xsd:decimal"/>
      <xsd:element name="Street" type="xsd:string"/>
      <xsd:element name="City"   type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
```

(to be continued)

Complex Type Extension *(2)*

```
  <xsd:complexType name="AustralianAddress">
    <xsd:complexContent>
      <xsd:extension base="PO:Address">
        <xsd:sequence>
          <xsd:element name="State" type="xsd:string"/>
          <xsd:element name="PostalCode" type="xsd:decimal"/>
          <xsd:element name="Country" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

</xsd:schema>
```

Complex Type Restriction

- Complex types may be restricted by eliminating or restricting attributes, and subsetting content models
- When restriction is used, instances of the derived type will always be valid for the base type as well

```
<!-- Uses the data type declarations from previous example -->
  <xsd:complexType name="AustralianPostalAddress">
    <xsd:complexContent>
      <xsd:restriction base="PO:AustralianAddress">
        <xsd:sequence>
          <xsd:element name="Number" type="xsd:decimal"/>
          <xsd:element name="Street" type="xsd:string"/>
          <xsd:element name="City" type="xsd:string" minOccurs="0"
                                                     maxOccurs="0"/>
          <xsd:element name="State" type="xsd:string"/>
          <xsd:element name="PostalCode" type="xsd:decimal"/>
          <xsd:element name="Country" type="xsd:string"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
```

Polymorphism *(1)*

- In XML schema, derived types can be used polymorphically with elements of the base type

```
<!-- Uses the data type declarations from previous example -->
  <xsd:complexType name="PurchaseOrder">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="shippingAddress" type="PO:Address"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:element name="billingAddress"  type="PO:Address"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:choice minOccurs="1" maxOccurs="1">
        <xsd:element name="BillingDate"  type="xsd:date"/>
        <xsd:element name="ShippingDate" type="xsd:date"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
```

Polymorphism *(2)*

- Using polymorphism in the XML schema instance

```
<?xml version="1.0" encoding="UTF-8">
<PO:PurchaseOrder xmlns:PO="http://www.auto-parts.com/PurchaseOrder">
  <Name> Auto Parts and Car Accessories </Name>
  <shippingAddress xsi:type="PO:AustralianAddress">
    <Number> 459 </Number>
    <Street> Wickham St. </Street>
    <City> Fortitude Valley </City>
    <State> QLD </State>
    <PostalCode> 4006 </PostalCode>
    <Country> Australia </Country>
  </shippingAddress>
  <billingAddress xsi:type="PO:AustralianPostalAddress">
    <Number> 158 </Number>
    <Street> Edward St. </Street>
    <State> QLD </State>
    <PostalCode> 4000 </PostalCode>
    <Country> Australia </Country>
  </billingAddress>
  <BillingDate> 2010-09-15 </BillingDate>
</PO:PurchaseOrder>
```

Including Schema *(1)*

- The **`include`** element allows for modularization of schema documents by including other schema documents in a schema document that has the same target namespace
- The included schema must be in the same or no namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:PO="http://www.auto-parts.com/PurchaseOrder"
  targetNamespace="http://www.auto-parts.com/PurchaseOrder">
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Address" type="PO:AddressType"
                   minOccurs="1" maxOccurs="1"/>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Including Schema *(2)*

- Using the **`include`** statement in the purchase order schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:PO="http://www.auto-parts.com/PurchaseOrder"
  targetNamespace="http://www.auto-parts.com/PurchaseOrder">
  <xsd:include
    schemaLocation="http://www.auto-parts.com/customerType.xsd"/>
  <xsd:include
    schemaLocation="http://www.auto-parts.com/productType.xsd"/>
  <xsd:element name="PurchaseOrder" type="PO:PurchaseOrderType"/>
  <xsd:complexType name="PurchaseOrderType">
    <xsd:all>
      <xsd:element name="ShippingInformation" type="PO:Customer"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:element name="BillingInformation" type="PO:Customer"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:element name="Order" type="PO:OrderType"
                   minOccurs="1" maxOccurs="1"/>
    </xsd:all>
  </xsd:complexType>
```

(to be continued)

Including Schema *(3)*

```
  <xsd:complexType name="AddressType">
    <xsd:sequence>
      <xsd:element name="Street"     type="xsd:string"/>
      <xsd:element name="City"       type="xsd:string"/>
      <xsd:element name="State"      type="xsd:string"/>
      <xsd:element name="PostalCode" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="Product" type="PO:ProductType"
                   minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
    ...
  </xsd:complexType>
</xsd:schema>
```

Importing Schema *(1)*

- The **`import`** element is used to import schema definitions from a different XML namespace; W3C XML Schema does not allow a single schema file to contain definitions for more than a single namespace
- The **`import`** element differs from the **`include`** element in two ways
  - The **`include`** element can only be used within the same namespace, while the **`import`** element is used across namespaces
  - The purpose of the **`include`** element is specifically to introduce other schema docs, while the purpose of the **`import`** element is to record dependency on another namespace, not necessarily another schema document
- The address schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:addr="http://www.auto-parts.com/Address"
  targetNamespace="http://www.auto-parts.com/Address">
  <xsd:complexType name="AddressType" abstract="true">
    <xsd:sequence>
      <xsd:element name="Number" type="xsd:decimal"/>
      <xsd:element name="Street" type="xsd:string"/>
```

(to be continued)

Importing Schema *(2)*

```
      <xsd:element name="City"   type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="AustralianAddress">
    <xsd:complexContent>
      <xsd:extension base="addr:AddressType">
        <xsd:sequence> ... </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="AustralianPostalAddress">
    <xsd:complexContent>
      <xsd:restriction base="addr:AustralianAddress">
        <xsd:sequence> ... </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Importing Schema *(3)*

- Using the **`import`** and **`include`** statements in the purchase order schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.auto-parts.com/PurchaseOrder"
  xmlns:PO="http://www.auto-parts.com/PurchaseOrder"
  xmlns:addr="http://www.auto-parts.com/Address">
  <xsd:include
    schemaLocation="http://www.auto-parts.com/productType.xsd"/>
  <xsd:import namespace="http://www.auto-parts.com/Address"
    schemaLocation="http://www.auto-parts.com/addressType.xsd"/>
  <xsd:element name="PurchaseOrder" type="PO:PurchaseOrderType"/>
  <xsd:complexType name="PurchaseOrderType">
    <xsd:all>
      <xsd:element name="ShippingInformation" type="PO:Customer"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:element name="BillingInformation" type="PO:Customer"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:element name="Order" type="OrderType"
                   minOccurs="1" maxOccurs="1"/>
    </xsd:all>
  </xsd:complexType>
```

(to be continued)

Importing Schema *(4)*

```
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Address" type="addr:AddressType"
                   minOccurs="1" maxOccurs="1"/>
      ...
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="Product" type="PO:ProductType"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
    ...
  </xsd:complexType>
</xsd:schema>
```

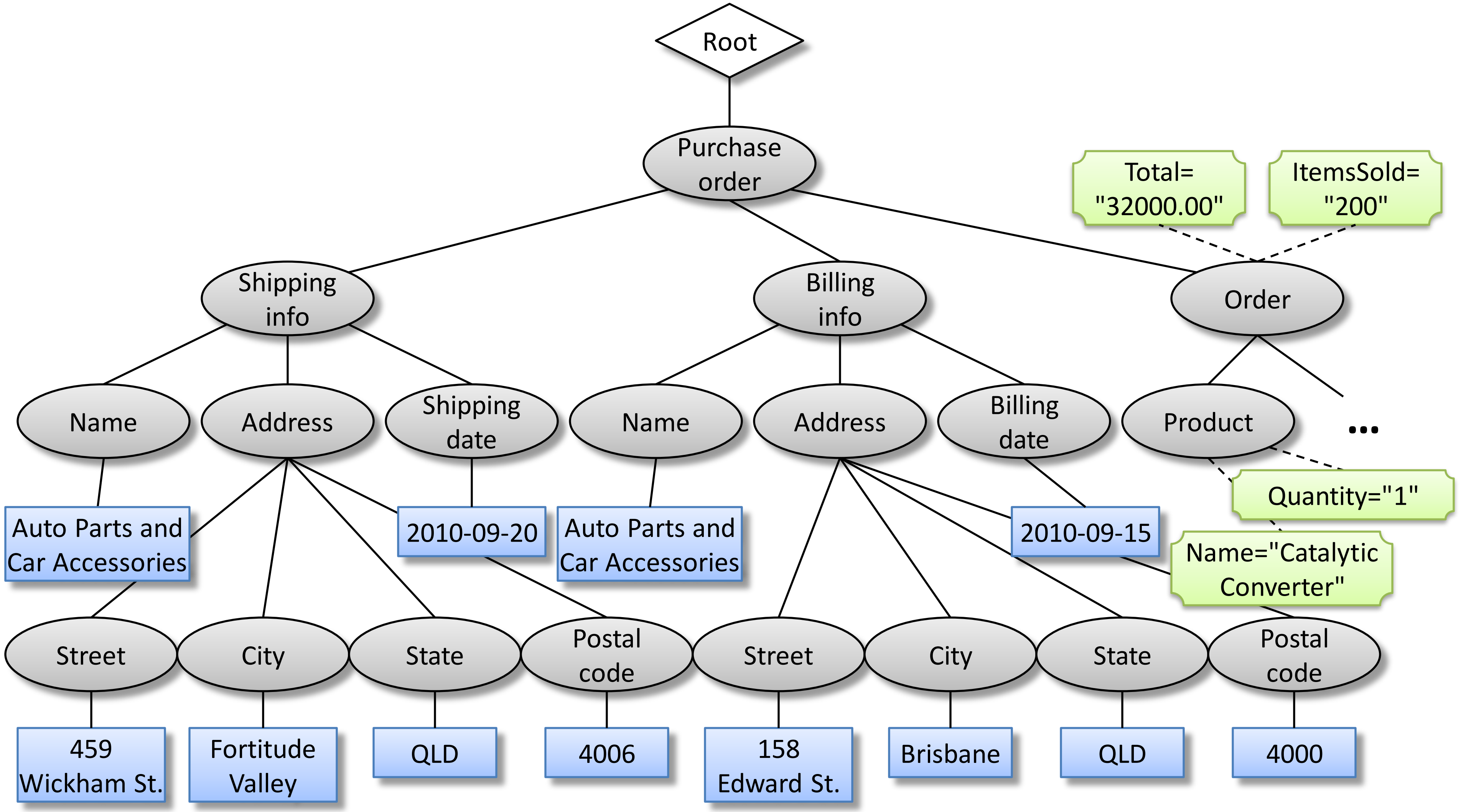
Document Navigation *(1)*

- XPath (XML Path Language) data model views a document as a tree of nodes; nodes correspond to document components such as elements and attributes
- XPath uses genealogical taxonomy to describe the hierarchical makeup of an XML document, referring to children, descendants, parents, and ancestors
  - The parent is the element that contains the element under discussion
  - A list of ancestors includes the parent of an element and the entire set of nodes preceding the parent in a directed path leading up to the root
  - A list of descendants includes the children of an element in a direct path all the way down to leaf nodes
- The topmost node in XPath is the *root* or *document root*; it is not an element but rather a logical construct that holds together the entire XML document

Document Navigation *(2)*

- The root element (or, the *document element*) – the child of the root – is the single element from which all other elements in the XML document are children or descendants; it is the first element in a document
- Nodes in an XPath tree are arranged in the following order:
  - The document order of nodes is based on the tree hierarchy of the XML instance
  - Element nodes are ordered prior to their children (to which they are connected via solid lines)
  - Children nodes of a given element are processed prior to sibling nodes
  - Attributes and namespaces (connected via dashed lines) of a given element are ordered prior to the children of the element

Example of XPath Tree



Document Navigation *(3)*

- Below is a sample XPath expression and the resulting node set

```
XPath Query#1: /PurchaseOrder/Order/child::*[1]
Resulting Node Set#1:
=====================
    <Product Name="Catalytic Converter" Price="240.00"
    Quantity="1"/>
```

- The XPath query consists of three location steps:
  - **`/PurchaseOrder`** $\rightarrow$ select element **`PurchaseOrder`** where the current context node is the root of the XML document
  - **`Order`** $\rightarrow$ select element **`Order`** where the context is the node **`PurchaseOrder`**
  - **`child::*[1]`** $\rightarrow$ select the first child element where the context is the node **`Order`**

Document Transformation

- XSLT (eXtensible Stylesheet Language Transform) can be used to format or transform XML content for presentation to users
- The *style sheet*, written in XSLT, specifies how the XML data will be displayed
- The converted document can be another XML document or a document in another format, such as HTML
- XSLT intensively uses XPath to address and locate sections of XML documents
- XSLT transformations are very useful for business applications, e.g., XML documents generated and used internally by an enterprise may need to be transformed into an equivalent format that customers or service providers of the enterprise are more familiar with

Example of XSLT Transformation

```
<BillingInformation>
  <Name> Plastic Products </Name>
  <Address>
    <Street> 158 Edward St. </Street>
    <State> QLD </State>
    <PostalCode> 4000 </PostalCode>
    <Country> Australia </Country>
  </Address>
</BillingInformation>
```

Source XML Application

Transformation service

Target XML Application

```
<BillingInfo>
  <Name> Plastic Products </Name>
  <BillingAddress>
    <Street> 158 Edward St. </Street>
    <PostalCode> QLD 4000 </PostalCode>
    <Country> Australia </Country>
  </BillingAddress>
</BillingInfo>
```