Business Process Digitalization and Cloud Computing

6. Service Context and Common Semantics

Andrea Morichetta, Phd

October 11, 2016

Computer Science Division

Table of contents

The importance of semantics in SOA

Semantics overview

- **Message semantics** is the most important requirement for service interoperability.
- This ensures that service consumers and providers exchange data in a consistent way

What we are going to see?

- How to **synthesize a model** by exposing details about a problem
- How to model a domain in terms of **objects, attributes, and associations**
- How to **partition** large models
- The usage of **XML** for representing these models

Semantics interoperability levels

- **Project-specific interoperability** lowest level of interoperability. Involves specific data formats creation for a particular SOA projects. Communication with other projects is possible only by means of transformation.
- **Business domain-specific interoperability** Involves reuse of data standards within a business domain. Projects can reuse message formats and can, therefore, interoperate with other services and consumers within that business domain.
- **Business domain-independent interoperability** data formats use standards from multiple business domains

Component of information modeling

Information model defines the **data** and **domain** concepts that must be shared between services.

- To understand a domain, you need to understand the things in the **domain** (the objects) and their **semantics** (their meaning, rules, and policies).

Objects

An object is defined as an abstraction of a set of things in a domain such that:

- All the things in the set have the **same characteristics**
- All the instances are subject to and conform to the same **behavior, rules, and policies**.

Objects and Attributes

Class incorporate things with common characteristics and common behavior.

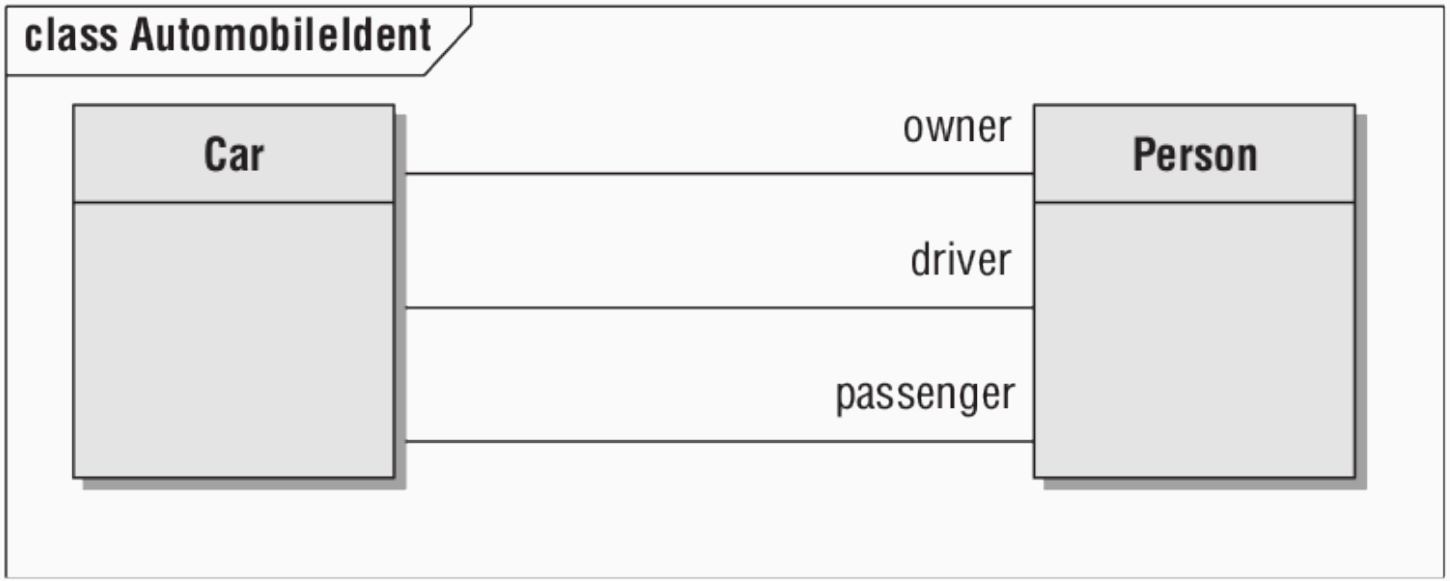
Attributes abstract the common characteristics of a class. Each attribute is:

- **relevant** for every instance of the class
- expected to have at most one value per instance

An Association is a relation between things. The association are formalized as **association**. Classes can have more than one association with different meaning

Association Multiplicities

Association Multiplicities specify for a given instance how many related instances of the other class can exist given the fact that the two classes are related. (zero, one, many)



Types

Every **data** consumed or produced by a services are formalized as **data types**.

Defining a data type it is possible to ensure the overall **accuracy** and **consistency** of the information model.

Domain specific data types represent the data types that typically compose the core concepts of a particular domain. There are three main basic categories:

1. **simple**: types that represent a single value
2. **composite**: single value that can be meaningfully subdivided into component
3. **document**: sophisticated data built of simple and composite types, typically organized into hierarchy.

Simple Types

Single atomic values that can be classified in:

- Numeric: can be defined as unit of measure, quantities, values, times, dates.
  The definition of a type includes both the **structure** of the type and the sets of **operations** that are permitted between values of that type and other types.

```
Type A is  10..20   by 1
Type B is   0..max  by 1
Type C is  32..212  by 0.01
Type D is -100..100 by 10
```

Simple Types

- Symbolic Types: represent **labels** and **descriptive text**. Typical operations supported for symbolic types include **combining** (concatenation), **splitting** (substring), and **parsing** (splitting according to patterns or grammars).

```
NameString is any text

ZipCode is exactly 5 characters
PostalCode is between 3 and 12 characters
Password is at least 10 characters
ContainerCode is up to 6 characters
CommentString is up to 200 characters and can be null

["+" digit+] ["(" digit+ ")"] digit+ [space digit+]*
```

Simple Types

- Enumeration represent **discrete value** taken from some defined set.

  ```
  ContainerCondition is (Clean, Dirty, Damaged)
  OrderState is (Unpaid, Paid, Packed, Delivered)
  ```

Composite Types

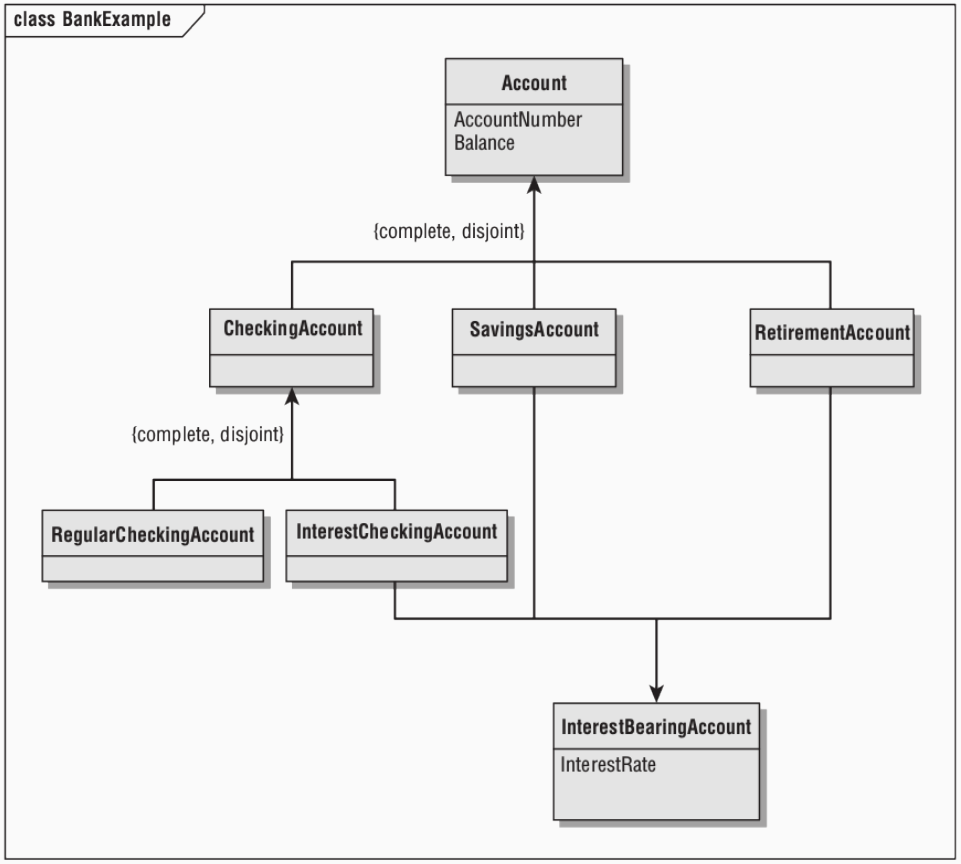
- Composite type are single atomic value that can contain several individual component.

```
Type Address is
    Street: string
    City: string
    State: UNSubdivisionCode
    PostalCode: PostalCode
    Country: ISOCountryCode
End Type
```

Identifiers

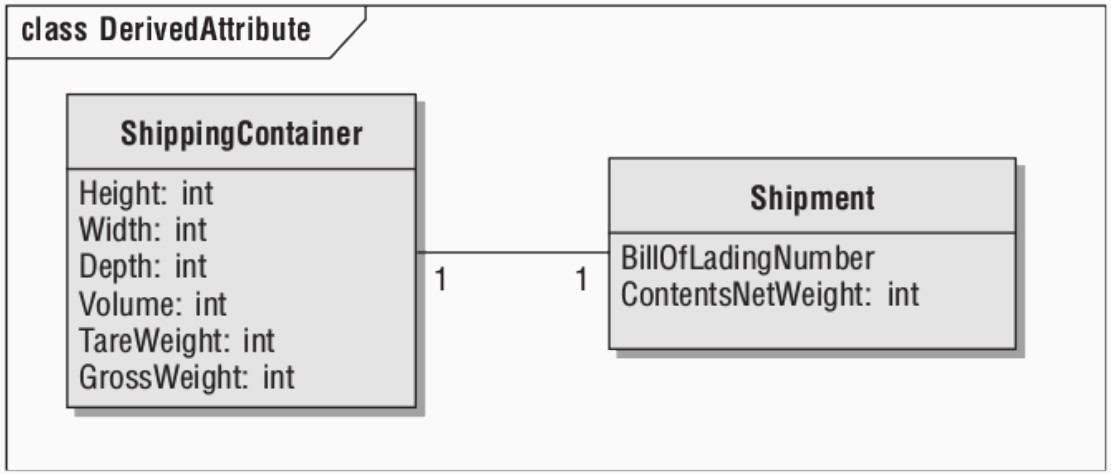
- An identifier is a collection of attributes that uniquely identifies an instance of an object (similar to the primary keys, defined in the entity/relational modeling, ISBN in the books context).
- A class is not required to have an identifier in the information model; identifiers serve to refer an object.
- Objects do not have natural identifiers, but it is still important to be able to identify instances. An attribute like *ChargeID* is a **contrived identifier**
- Subpopulation Identifiers: some attributes are unique but only in the context of an association to another class

Specialization



Specialization permit to model common attributes, associations, and behaviors in a **superclass** and then to model the different attributes, associations, and behaviors in separate **subclasses**.

Derived Attributes



- The values of derived attributes are **originated** from the values of other attributes in the model
- The semantic information model contains the information, including derived attributes, but **not the rules** or **formulas** that calculate them.

Documents based on the information model

Defining documents

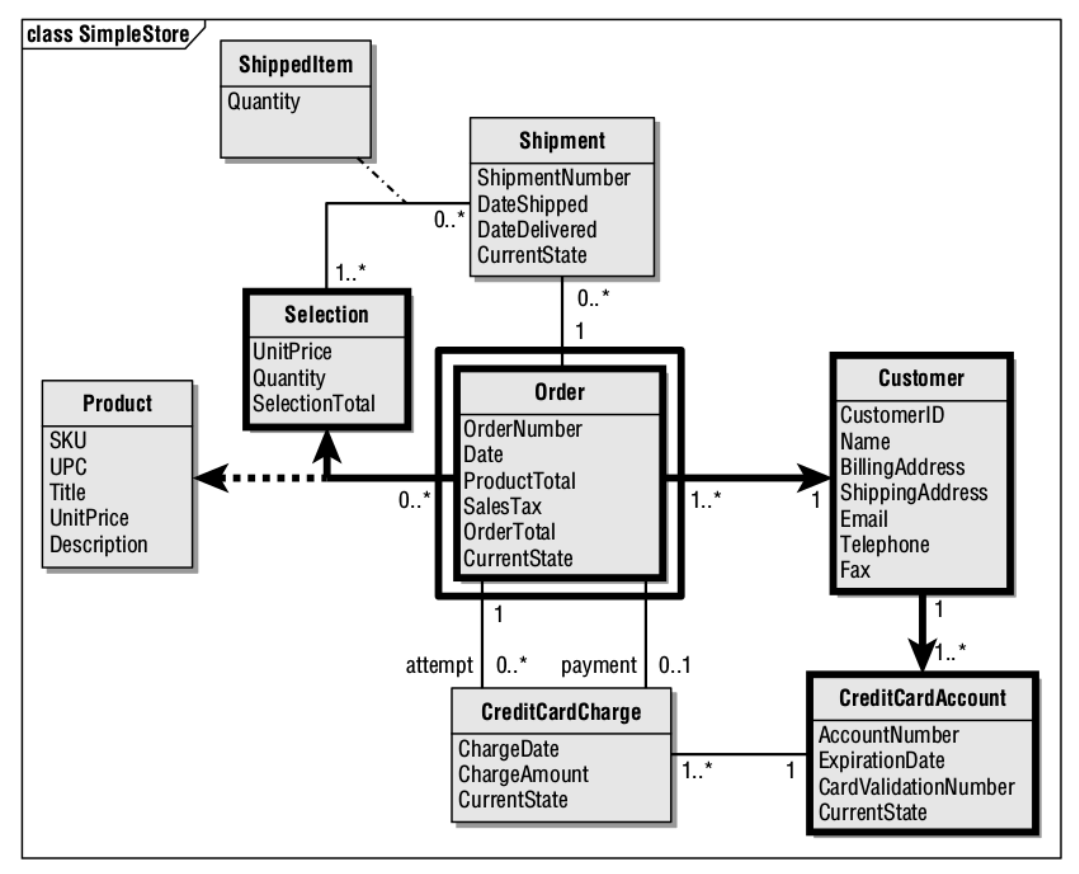
Documents are typically **containers of information**, specific for a given service

Documents enclose together **multiple domain objects** to provide input/output for a given service operation.

To define a document, is important to draw the **structure of the document** on top of the information model.

Documents

```
Order
  OrderNumber 2217843
  Date 12/15/2007
  ProductTotal $ 684.85
  SalesTax  $ 56.50
  OrderTotal  $ 741.35
  Selection
    UPC 0785357834163
    UnitPrice $45.99
    Quantity 2
  Selection
    UPC 9780201748048
    UnitPrice $44.99
    Quantity 5
  Customer
    Name  Samuel L. Clemens
    BillingAddress 1234 Tom Sawyer ...
    CreditCardAccount
      AccountNumber 9823-2132-7983
      ExpirationDate 2/2004
      CardValidationNumber 999
```



XML

In the majority of SOA implementations, the objects and documents that are exchanged are in XML, that is the **standard** de facto for **data messaging** in service implementations.

- Is a **standard syntax** for metadata and a **standard structure** for documents.
- It is **independent of programming languages** and operating environment
- Programming language provides good support for **marshaling/unmarshaling** XML payloads.
- It is, **extensible**, and its extensibility makes it easier to support changes
- it is an **open standard**, accepted by industry and the major vendors

XML

```
<Order>
  <OrderNumber>2217843</OrderNumber>
  <Date>12/15/2007</Date>
  <ProductTotal>684.85</ProductTotal>
  <SalesTax>56.50</SalesTax>
  <OrderTotal>741.35</OrderTotal>
  <Selection>
     <UPC>0785357834163</UPC>
     <UnitPrice>45.99</UnitPrice>
     <Quantity>2</Quantity>
  </Selection>
  <Selection>
     <UPC>9780201748048</UPC>
     <UnitPrice>44.99</UnitPrice>
     <Quantity>5</Quantity>
  </Selection>
  <Customer>
     <Name>
        <FirstName>Samuel</FirstName>
        <MiddleInitial>L</MiddleInitial>
        <LastName>Clemens</LastName>
     </Name>
     <BillingAddress>
        <Street>1234 Tom Sawyer Drive</Street>
        <City>Hannibal</City>
        <State>MO</State>
        <Zip>63401</Zip>
     </BillingAddress>
     <CreditCardAccount>
        <AccountNumber>9823-2132-7983</AccountNumber>
        <ExpirationDate>2/2004</ExpirationDate>
        <CardValidationNumber>999</CardValidationNumber>
     </CreditCardAccount>
  </Customer>
</Order>
```

XML Schema

- XML schema is a **definition language** that enables to constrain XML documents to a specific vocabulary and hierarchical structure.
- XML documents can be **validated against a schema**, and this validation process can catch many **structural** and **semantic** errors in the document.

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the **elements and attributes** that can appear in a document
- the **number** of (and **order** of) child **elements**
- **data types** for elements and attributes
- **default and fixed values** for elements and attributes

XML Schema example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Order">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="OrderNumber" type="xs:int"/>
    <xs:element name="Date" type="xs:date"/>
    <xs:element name="ProductTotal" type="xs:decimal"/>
    <xs:element name="SalesTax" type="xs:decimal"/>
    <xs:element name="OrderTotal" type="xs:decimal"/>
    <xs:element name="Selection" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="UPC" type="xs:long"/>
       <xs:element name="UnitPrice" type="xs:decimal"/>
       <xs:element name="Quantity" type="xs:int"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
 <xs:element name="Customer">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Name">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="FirstName" type="xs:string"/>
       <xs:element name="MiddleInitial" type="xs:string"/>
       <xs:element name="LastName" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
 <xs:element name="BillingAddress">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Street" type="xs:string"/>
    <xs:element name="City" type="xs:string"/>
    <xs:element name="State" type="xs:string"/>
    <xs:element name="Zip" type="xs:int"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="CreditCardAccount">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="AccountNumber"
                type="xs:string"/>
    <xs:element name="ExpirationDate"
                type="xs:gYearMonth"/>
    <xs:element name="CardValidationNumber"
                type="xs:short"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

The XSD Document

- Since the XSD is written in XML, it can get confusing which we are talking about
- The file extension is **.xsd**
- The root element is <schema>
- The XSD starts like this:

```
<?xml version="1.0"?>
 <xs:schema xmlns:xs="http://www.w3.rg/2001/XMLSchema">
targetNamespace="http://www.w3schools.com"
 xmlns="http://www.w3schools.com"
 elementFormDefault="qualified">
```

The XSD details

xmlns:xs="http://www.w3.org/2001/XMLSchema"
indicates that the **elements and data types** used in the schema come from the www.w3.org... namespace.

The namespace should be prefixed with **xs:**

xmlns="http://www.w3schools.com"
indicates that the default namespace

elementFormDefault="qualified"
indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a schema

To refer to an XML Schema in an XML document, **the reference goes in the root element**:

***.xml

```
<?xml version="1.0"?>
<rootElement
  <!--The XML Schema Instance reference is required-->
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <!-- XML Schema path-->
  xsi:noNamespaceSchemaLocation="url.xsd">

    ...
</rootElement>
```

Simple and Complex elements

- A **simple** element is one that contains text and nothing else
  - A simple element cannot have attributes
  - A simple element cannot contain other elements
  - A simple element cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- If an element isn't simple, it's **complex**
  - A complex element may have attributes
  - A complex element may be empty, or it may contain text, other elements, or both text and other elements

Defining Simple element

- A **simple element** is defined as

```
<xs:element name="name" type="type" />
```

  Where:

  - **name** is the name of the element
  - the most common **values for type** are:

xs:boolean	xs:integer
xs:date	xs:string
xs:decimal	xs:time

- Other attributes a simple element may have:
  - **default="default value"** if no other value is specified
  - **fixed="value"** no other value may be specified

Defining an attribute

If an element has **attributes**, it is considered to be of a complex type

- Attributes themselves are always declared as simple types
- An attribute is defined as:

  ```
  <xs:attribute name="name" type="type" />
  ```

  where:

  name and **type** are the same as for **xs:element**
- Other attributes a simple element may have:
  - **default="default value"** if no other value is specified
  - **fixed="value"** no other value may be specified
  - **use="optional"** the attribute is not required (default)
  - use="**required**" the attribute is mandatory

```
<xs:attribute name="lang" type="xs:string"
              default="EN" use="required"/>
```

Restrictions on contents

Restrictions are used to define acceptable values for XML elements or attributes.

- The general form for putting a restriction on a text value is:

```
<xs:element name="name">
   <xs:restriction base="type">
      ... the restrictions ...
   </xs:restriction>
</xs:element>
```

- For example:

```
<xs:element name="age">
   <xs:restriction base="xs:integer">
      <xs:minInclusive value="0">
      <xs:maxInclusive value="140">
   </xs:restriction>
</xs:element>
```

Restrictions on numbers

- **minInclusive** – number must be $\geq$ the given value
- **minExclusive** – number must be $>$ the given value
- **maxInclusive** – number must be $\leq$ the given value
- **maxExclusive** – number must be $<$ the given value
- **totalDigits** – number must have **exactly** value digits
- **fractionDigits** – number must have **no more than** value digits after the decimal point

Restrictions on strings

- **length** – the string must contain exactly value characters
- **minLength** – the string must contain at least value characters
- **maxLength** – the string must contain no more than value characters
- **pattern** – the value is a regular expression that the string must match

  ```
  <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
  ```

- **whiteSpace** – not really a restriction–tells what to do with whitespace
  - value="**preserve**" Keep all whitespace
  - value="**replace**" Change all whitespace characters to spaces
  - value="**collapse**" Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration

- An enumeration **restricts the value** to be one of a fixed set of values
- Example:

```
<xs:element name="season">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="Spring"/>
         <xs:enumeration value="Summer"/>
         <xs:enumeration value="Autumn"/>
         <xs:enumeration value="Winter"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Complex Elements

A complex element is an XML element that contains **other elements and/or attributes**.

There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Complex Element definition 1

- **Schema**

```
<xs:element name="employee">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="firstname" type="xs:string"/>
         <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

- **Definition**

```
<employee>
   <firstname>John</firstname>
   <lastname>Smith</lastname>
</employee>
```

Complex Element definition 2

- **Schema**

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

ComplexType element can have **name**, and other elements can refers to the name of this complexType (using this method, **several elements can refer to the same complex type**):

Sequence Indicator

```
<xs:element name="person">
  <xs:complexType>
     <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>
```

The <**xs:sequence**> tag means that the elements defined ("firstname" and "lastname") must appear in that order inside an element.

All indicator

```
<xs:element name="person">
   <xs:complexType>
      <xs:all>
         <xs:element name="firstName" type="xs:string" />
         <xs:element name="lastName" type="xs:string" />
      </xs:all>
   </xs:complexType>
</xs:element>
```

- <xs:all> allows **elements to appear in any order**
- Despite the name, the members of an xs:all group can occur once or not at all
- You can use minOccurs="0" to specify that an element is optional (default value is 1)

Choice Indicator

The <choice> indicator specifies that **either** one child element or another can occur

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

minOccurs/maxOccurs Indicator

- The minOccurs indicator specifies the minimum number of times an element can occur
- The maxOccurs indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
          maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Empty element

An empty complex element cannot have contents, only attributes

- **Schema**

```
<xs:element name="product">
   <xs:complexType>
      <xs:attribute name="prodid" type="xs:int"/>
   </xs:complexType>
</xs:element>
```

- **Definition**

```
<product prodid="1345" />
```

Mixed element

- Mixed elements may contain **both text and elements**
- We add mixed="true" to the **xs:complexType** element
- The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInt"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<letter>
 Dear Mr.<name>John Smith</name>.
 Your order <orderid>1032</orderid>
 will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

Extensions

- XML file can contains components from two different schemas

```
<persons xmlns="http://www.microsoft.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.microsoft.com family.xsd
  http://www.w3schools.com children.xsd">
```

- The any and anyAttribute elements are used to make EXTENSIBLE documents! They allow documents to contain additional elements that are not declared in the main XML schema.

```
<xs:any minOccurs="0"/>
  or
<xs:anyAttribute/>
```

XSD Data type

XSD String

- Recall that a simple element is defined as:

```
<xs:element name="name" type="type" />
```

- Here are a few of the possible string types:
  - **xs:string** – a string
  - **xs:normalizedString** – a string that doesn't contain tabs, newlines, or carriage returns
  - **xs:token** – a string that doesn't contain any whitespace other than single spaces
- Allowable **restrictions** on strings: enumeration, length, maxLength, minLength, pattern, whiteSpace

XSD Date

- **xs:date** – A date in the format **CCYY-MM-DD**, for example, 2002-11-05
- **xs:time** – A date in the format **hh:mm:ss** (hours, minutes, seconds)
- **xs:dateTime** – Format is **CCYY-MM-DDThh:mm:ss** The T is part of the syntax
- Allowable **restrictions** on dates and times: enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

XSD Numeric

- Predefined numeric data types:

  xs:decimal
  xs:byte
  xs:short
  xs:int
  xs:long
  xs:positiveInteger
  xs:negativeInteger
  xs:nonPositiveInteger
  xs:nonNegativeInteger

- Allowable **restrictions** on numeric types: enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

XSD Misc

- **Boolean Data Type**

```
<xs:attribute name="disabled" type="xs:boolean"/>
```

- **Binary Data Types**:
  base64Binary (Base64-encoded binary data)
  hexBinary (hexadecimal-encoded binary data)

```
<xs:element name="blobsrc" type="xs:hexBinary"/>
```

- **AnyURI Data Type**

```
<xs:attribute name="src" type="xs:anyURI"/>
<pic src="http://www.google.com" />
```

- Allowable **restrictions** on Miscellaneous Data Types:
  enumeration (a Boolean data type cannot use this constraint)
  length, maxLength, minLength (a Boolean data type cannot use this constraints), pattern, whiteSpace

XML validator

Questions?