أكاديمية الحكومة الإلكترونية الفلسطينية

The Palestinian eGovernment Academy

www.egovacademy.ps

Tutorial III:

Process Integration and Service Oriented Architectures

# Session 2
# Overview XML NS and Schema


Prepared By

*Mohammed Aldasht*

Reviewed by

Prof. Marco Ronchetti and Prof. Paolo Bouquet, Trento University, Italy



European Commission

TEMPUS

# About

This tutorial is part of the PalGov project, funded by the TEMPUS IV program of the Commission of the European Communities, grant agreement 511159-TEMPUS-1-2010-1-PS-TEMPUS-JPHES. The project website: www.egovacademy.ps

## Project Consortium:

- Birzeit University, Palestine (**Coordinator** )
- Palestine Polytechnic University, Palestine
- Palestine Technical University, Palestine
- Ministry of Telecom and IT, Palestine
- Ministry of Interior, Palestine
- Ministry of Local Government, Palestine
- University of Trento, Italy
- Vrije Universiteit Brussel, Belgium
- Université de Savoie, France
- University of Namur, Belgium
- TrueTrust, UK


**Coordinator:**
Dr. Mustafa Jarrar
Birzeit University, P.O.Box 14- Birzeit, Palestine
Telfax:+972 2 2982935 mjarrar@birzeit.edu

# © Copyright Notes

Everyone is encouraged to use this material, or part of it, but should properly cite the project (logo and website), and the author of that part.

No part of this tutorial may be reproduced or modified in any form or by any means, without prior written permission from the project, who have the full copyrights on the material.


**Attribution-NonCommercial-ShareAlike**
**CC-BY-NC-SA**

**This license lets others remix, tweak, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms.**

# Tutorial Map

| Intended Learning Objectives |
|---|
| **A: Knowledge and Understanding** |
| 3a1: Demonstrate knowledge of the fundamentals of middleware. |
| 3a2: Describe the concept behind web service protocols. |
| 3a3: Explain the concept of service oriented architecture. |
| 3a4: Explain the concept of enterprise service bus. |
| 3a5: Understanding WSDL service interfaces in UDDI. |
| **B: Intellectual Skills** |
| 3b1: Design, develop, and deploy applications based on Service Oriented Architecture (SOA). |
| 3b2: use Business Process Execution Language (BPEL). |
| 3b3: using WSDL to describe web services. |
| **C: Professional and Practical Skills** |
| 3c1: setup, Invoke, and deploy web services using integrated development environment. |
| 3c2: construct and use REST and SOAP messages for web services communication. |
| **D: General and Transferable Skills** |
| d1: Working with team. |
| d2: Presenting and defending ideas. |
| d3: Use of creativity and innovation in problem solving. |
| d4: Develop communication skills and logical reasoning abilities. |

| Title | T | Name |
|---|---|---|
| Session0: Syllabus and overview | 0 | Aldasht |
| Sesson1: Introduction to SOA | 2 | Aldasht |
| Session2: XML namespaces & XML schema | 2 | Aldasht |
| Session 3: Xpath & Xquery | 4 | Romi |
| Session4: REST web services | 3 | M. Melhem |
| Session5: Lab2: Practice on REST | 3 | M. Melhem |
| Session 6: SOAP | 2 | Aldasht |
| Session 7: WSDL | **3** | Aldasht |
| Session8: Lab 3: WSDL practice | 3 | Aldasht |
| Session9: ESB | 4 | Aldasht |
| Session10: Lab4: Practice on ESB | 4 | Aldasht |
| Session11: integration patterns | 4 | M. Melhem |
| Session12: Lab5: integration patterns | 4 | M. Melhem |
| Session13: BPEL | 3 | Aldasht |
| Session14: Lab6: Practice on BPEL | 3 | Aldasht |
| Session15: UDDI | 2 | Aldasht |

# Session Outlines

- **XML Overview**
  - XML document and Grammars
- XML information set
- XML Namespaces
- XML Schema

# Overview

- XML is a markup language, like HTML, consists of *markup* and *text*.
- Markup is composed of individual tags.
- Both HTML and XML are languages for exchanging data, but there is a difference between them.
- See next slide for and initial XML example.

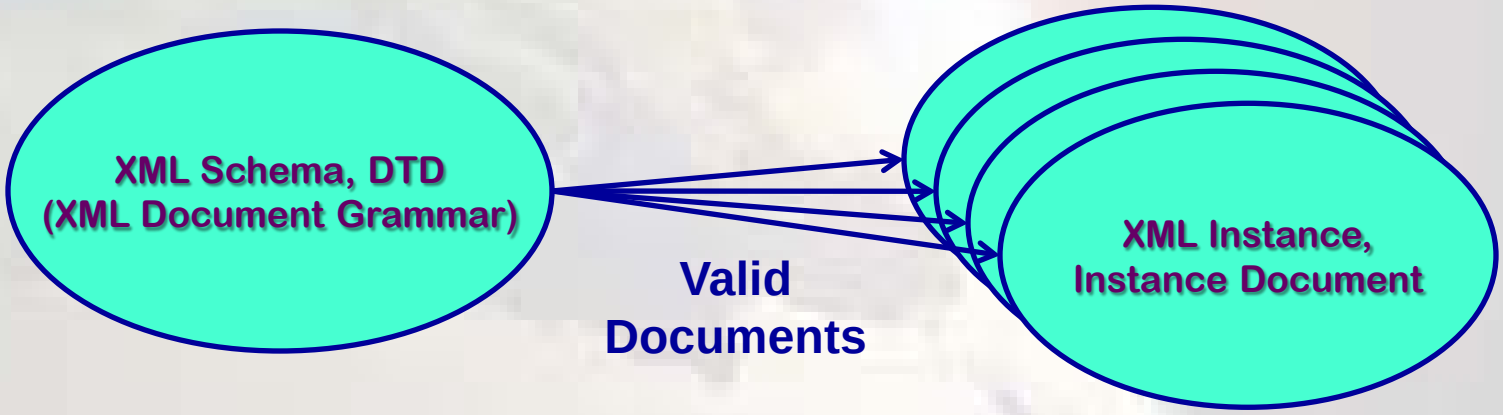
# Valid and well-formed XML document

- Valid XML document is one that comply with the constraints expressed through a given grammar.
- A well-formed XML document is one that comply with XML syntax expressed in the XML standard.
  - But not associated with a distinct grammar.

# XML grammars

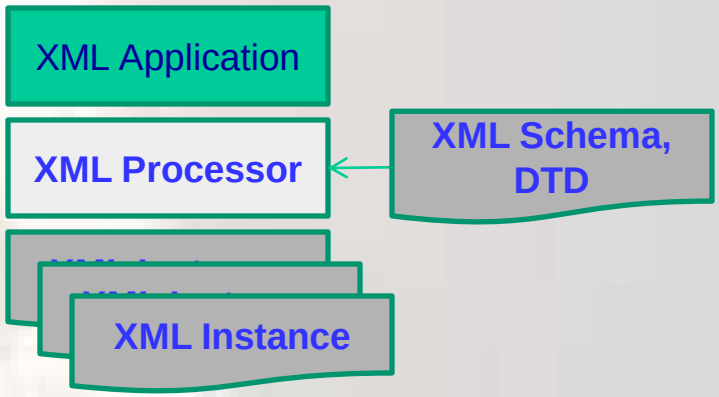
- Specified by two dominant concepts:
  - *XML schema* and *Document Type Definitions* (DTDs)
- XML schema is powerful to express structural XML document constraints than DTDs.
- XML document complying with a DTD or a schema is called and *XML instance* or *instance document*.
- See next slide, for the relationship between XML document grammar and XML instances resulting from applying the grammar.

# XML document grammar and valid XML instances, [2]



# XML processor

- Applications are interested in the structural information and the XML instance, when XML is used to exchange data between them.
- XML processor, e.g. SOAP server, must validate the XML documents against the XML grammar and pass the XML instance structure and payload to the application, e.g. SOAP message.




A validating XML processor, [2]

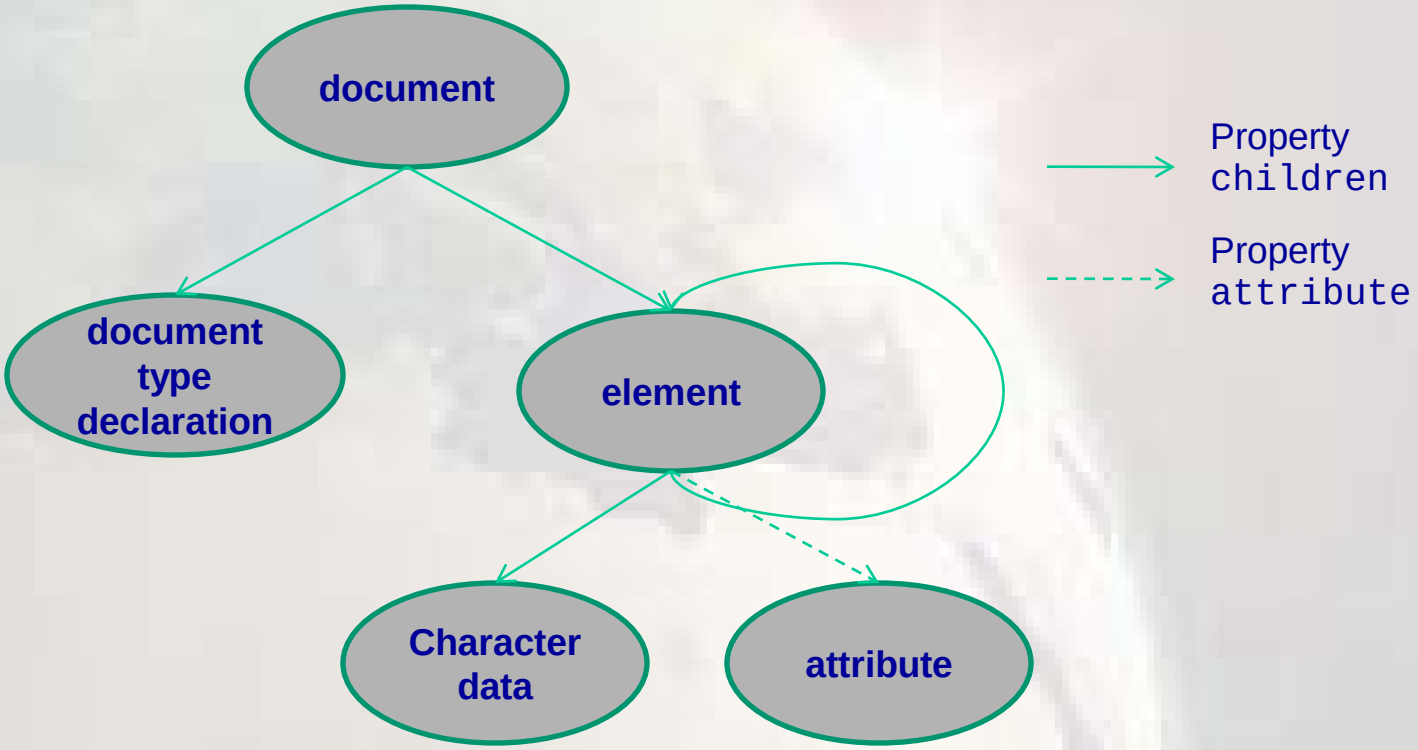
# Session Outlines

- XML Overview
  - XML document and Grammars
- **XML information set**
- XML Namespaces
- XML Schema

# XML information set

- XML info set, provides a set of abstract data definitions to represent the information in a well-formed XML document.
  - Each well-formed XML document has an associated info set.
- The information set consists of information items.
  - Each item describes an XML document part through a set of named properties

# Information item types of an XML information set, [2]



# Information item types of an XML information set

- `document`, consists at least of one mandatory *root element*, and:
  - XML version info (optional) & encoding info for the document. These are called *XML declaration*.
  - Document type declaration: contains markup declarations provide the grammar.
  - These together is called *prolog*.

# Information item types of an XML information set, cont.

- `element`: identified by a name and has a set of associated attributes.
- `attribute`: consists of a name and an associated value.
- `character data`: is an information item comprises the payload of an XML document.
- `comment`: `element` and `document` information items may contain comments.

# Information item types examples [2]

- Comment info item “may span multiple lines”:
  - `<!– This is a comment -->`
- Element info item:
  - element may have no content e.g.: `<address/>` or `<address></address>`

```
<address>
        <name>Mr Ahmad Ahmad</name>
        <street>11 Alquds Street</street>
        <city>Ramallah</city>
        <postal-code>100</postal-code>
        <country>Palestine</country>
</address>
```

# Information item types examples [2]

- An attribute is specified in the start tag of an element and consists of a name-value pair.
  - This example links an attribute named `targetAddress` with the "PS" to the `address` elements:

```
<address targetAddress="PS">
        <name>Mr Ahmad Ahmad</name>
        <street>11 Alquds Street</street>
        <city>Ramallah</city>
        <postal-code>100</postal-code>
        <country>Palestine</country>
</address>
```

# Information item types examples [2]

- Document info item: look to the following prolog:

```
<?xml version="1.0" encoding=UTF-16"?>
<! DOCTYPE address [<!– DTDs go here --> ]>
<address> <!– XML instances go here --> </address>
```

- Document type declaration identified by keyword DOCTYPE must be identical to the corresponding root element

# Information item types examples

- element content: if declared to be character data, this is indicated by the term #PCDATA. "*Parsed Character Data*"
  - Thus, a valid declaration would be:

```
<?xml version="1.0" encoding=UTF-16"?>
<! DOCTYPE address [<!ELEMENT address (#PCDATA)> ]>
```

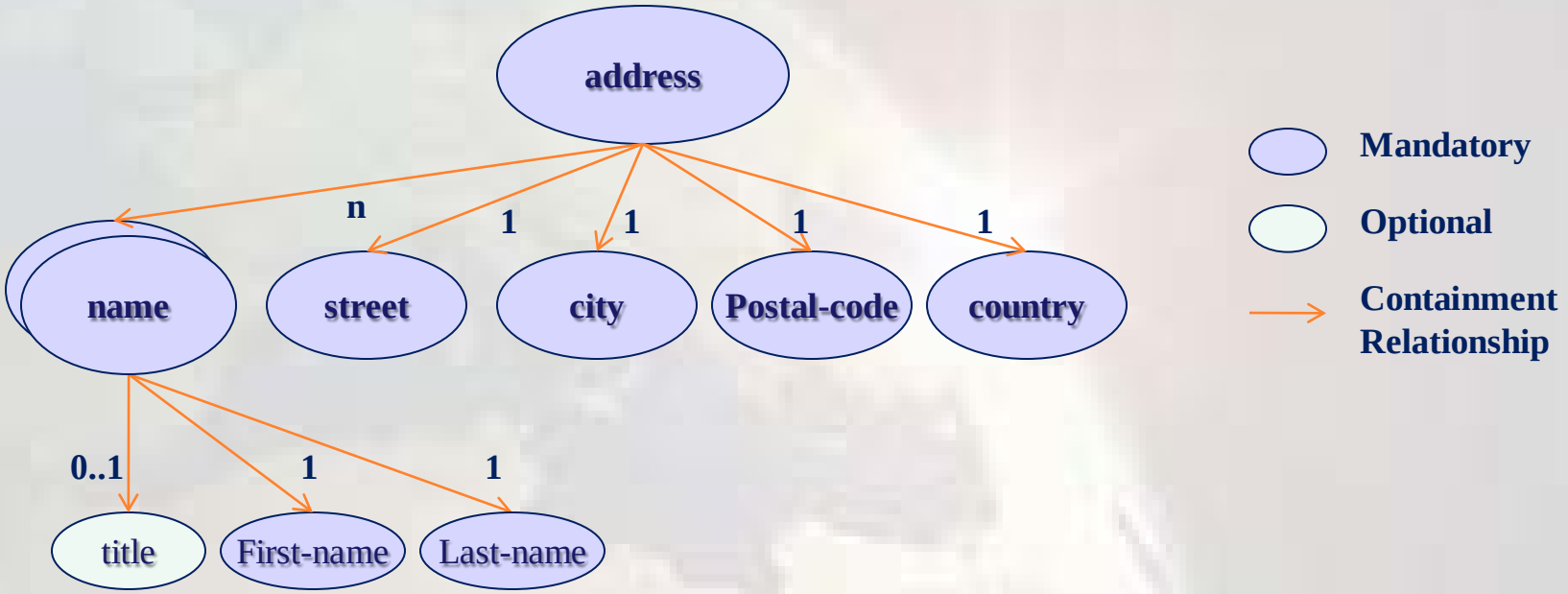
- In an XML instance the address element could appear as follows:

```
<address>
        Mr Ahmad Ahmad
        11 Alquds Street
        Ramallah
        100
        Palestine
</address>
```

# Element content and mixed content

- We can combine elements and build nested element declarations.
- DTD syntax provides 5 symbols used to describe manners of combination.
- Assume `e1`, `e2` and `e3` to be elements:
  - `e1?` : `?` means none or one element `e1`.
  - `e1*` : `*` means none, one or more element `e1`.
  - `e1+` : `+` means one or more than one element `e1`.
  - `e1, e2, e3` : `,` means list of element are chained.
  - `e1 | e2`: means `e1` or `e2` can be chosen but not both.

# address element can have this structure, [2]




Source, [2]

# A DTD describing the address structure, [2]

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE address [
<!ELEMENT address (name+, street, city, postal-code,
country)>
<!ELEMENT name (title?, first-name, last-name)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT postal-code(#PCDATA)>
<!ELEMENT country(#PCDATA)>
]>
```

# An instance document could be as shown here

```
<address>
        <name>
            <title selectTitle="Mr"/>
            <first-name>Ahmad M.</first-name>
            <last-name>Ahmad</last-name>
        </name>
        <street>11 Alquds Street</street>
        <city>Ramallah</city>
        <postal-code>100</postal-code>
        <country>Palestine</country>
</address>
```

# A valid address XML instance

```
<address targetAddress="PS">
        <name>
            <title selectTitle="Mr"/>
            <first-name>Ahmad M.</first-name>
            <last-name>Ahmad</last-name>
        </name>
        <street>11 Alquds Street</street>
        <city>Ramallah</city>
        <postal-code>100</postal-code>
        <country>Palestine</country>
</address>
```

# Session Outlines

- XML Overview
  - XML document and Grammars
- XML information set
- **XML Namespaces**
- XML Schema

# XML Namespaces

- XML processor must be able to differentiate our XML address instances from someone else's address instances.
  - Identification using element type names is not sufficient.
- Some global naming mechanism is required.
- XML *namespaces* ensure that XML definitions are unique.
- Using XML namespaces, XML elements can be distinguished, even if they have identical names.

# XML Namespaces concept

- An XML namespace comprises a collection of element type names & attribute names.
- An XML namespace, is identified by a URI reference.
- The collection of element type names & attribute names, belonging to the same namespace are identified by the namespace URI reference.
- See next slide for namespaces example!

# Identical markup belonging to different namespaces

http://companyx.com/ns/orders

<order>
<order-number>
<order-date>
<name>
<address>

http://companyx.com/ns/employees

<employee-id>
<dept-number>
<name>
<address>

http://clubx.com/ns/members

<member>
<member-id>
<member-since>
<name>
<address>

**Source, [2], with modifications**

# Qualified names

- A name from a namespace appears in a document as a *qualified name* (*Qname*).
- A Qname consists of a *prefix* and a *local part*.
- e.g. `cX` is the prefix and `address` is the local part:

  `cX:address`
- Prefix selects the namespace and local part take care of the naming within the scope of the namespace.

# Declaring XML namespaces

- Done through the reserved namespace attribute `xmlns`.
- Also, can be done through `xmlns:` followed by a name without colons.
- The value of the namespace attribute is the URI reference.
- Linking a namespace to a prefix, e.g.:

```
<address xmlns:cX="http://companyx.com/ns/employees">
```

  - `cX` is the prefix for all qualified names belonging to the namespace
- Using a default namespace, e.g.:

```
<address xmlns="http://companyx.com/ns/employees">
```

  - All subordinate elements are in the same default namespace, unless a subordinate element overwrites the default namespace.

# Declaring a namespace for the address document, using prefix namespace attribute

```
<cX:address xmlns:cX="http://companyx.com/ns/employees"
            targetAddress="PS">
        <cX:name>
            <cX:title selectTitle="Mr"/>
            <cX:first-name>Ahmad M.</cX:first-name>
            <cX:last-name>Ahmad</cX:last-name>
        </cX:name>
        <cX:street>11 Alquds Street</cX:street>
        <cX:city>Ramallah</cX:city>
        <cX:postal-code>100</cX:postal-code>
        <cX:country>Palestine</cX:country>
</cX:address>
```

# Using a default namespace name declaration

```
<address xmlns="http://companyx.com/ns/employees"
         targetAddress="PS">
      <name>
         <title selectTitle="Mr"/>
         <first-name>Ahmad M.</first-name>
         <last-name>Ahmad</last-name>
      </name>
      <street>11 Alquds Street</street>
      <city>Ramallah</city>
      <postal-code>100</postal-code>
      <country>Palestine</country>
</address>
```

# An example for an entry of the phone book maintained by the company

```
<phonebook xmlns="http://companyx.com/ns/phonebook"
      <location>Company X Office</location>
      <roomNumber>03.02</roomNumber>
      <!-- Extension -->
      <officePhone>*2911111</officePhone>
</phoneBook>
```

# XML namespaces and attributes

- Default namespaces are not applied to attributes that do not have a prefix.
- The following example is a valid XML instance.
  - Although the local part of the `location` element attribute is Identical.

```
<!-- Declare a namespace prefix
<phoneOwner xmlns:phone="http://companyx.com/ns/phonebook"
  <!-- for illustrative purposes, not a good XML practice. -->
  <phonebook xmlns="http://companyx.com/ns/phonebook"
      <location department="HR" phone:department="HR">
        Company X Office</location>
  </phoneBook>
</phoneOwner>
```

# Session Outlines

- XML Overview
  - XML document and Grammars
- XML information set
- XML Namespaces
- **XML Schema**

# XML schema

- If we want to insert the `roomNumber` element to the phone book as follows:
  - A `roomNumber` value must start with 2 digits describing the building floor, followed by a dot.
  - Then, a 2 more digits to represent the room number on that floor.
- No way to specify this pattern through a DTD.
- XML schema allows us to express such a constraints.
- Various data types can be defined with XML schema and new data types can be derived from existing ones.

# An initial XML schema example

**Type** definition

It is a simple type based on XML schema built-in type "string" restricting the string to a distinct string pattern (the string pattern is also called a *facet*)

```
<simpleType name="roomNumberType">
   <restriction base="string">
      <pattern value="[0-9]{2}\.[0-9]{2}"/>
   </restriction>
</simpleType>
```

```
<element name="roomNumber" type="roomNumberType"/>
```

**Element declaration**

The element named `roomNumber` is of type `roomNumberType`

**Source, [2]**

# The structure of an XML schema definition

- An *XML Schema Definition* (XSD) is itself an XML instance.
  - An advantage of the XML schema.
- The top element of the `xsd` is name `schema`.
- The XML namespace for a schema definition is `http://www.w3.org/2001/XMLSchema`, linked to prefix `xsd`. Start a schema definition as:

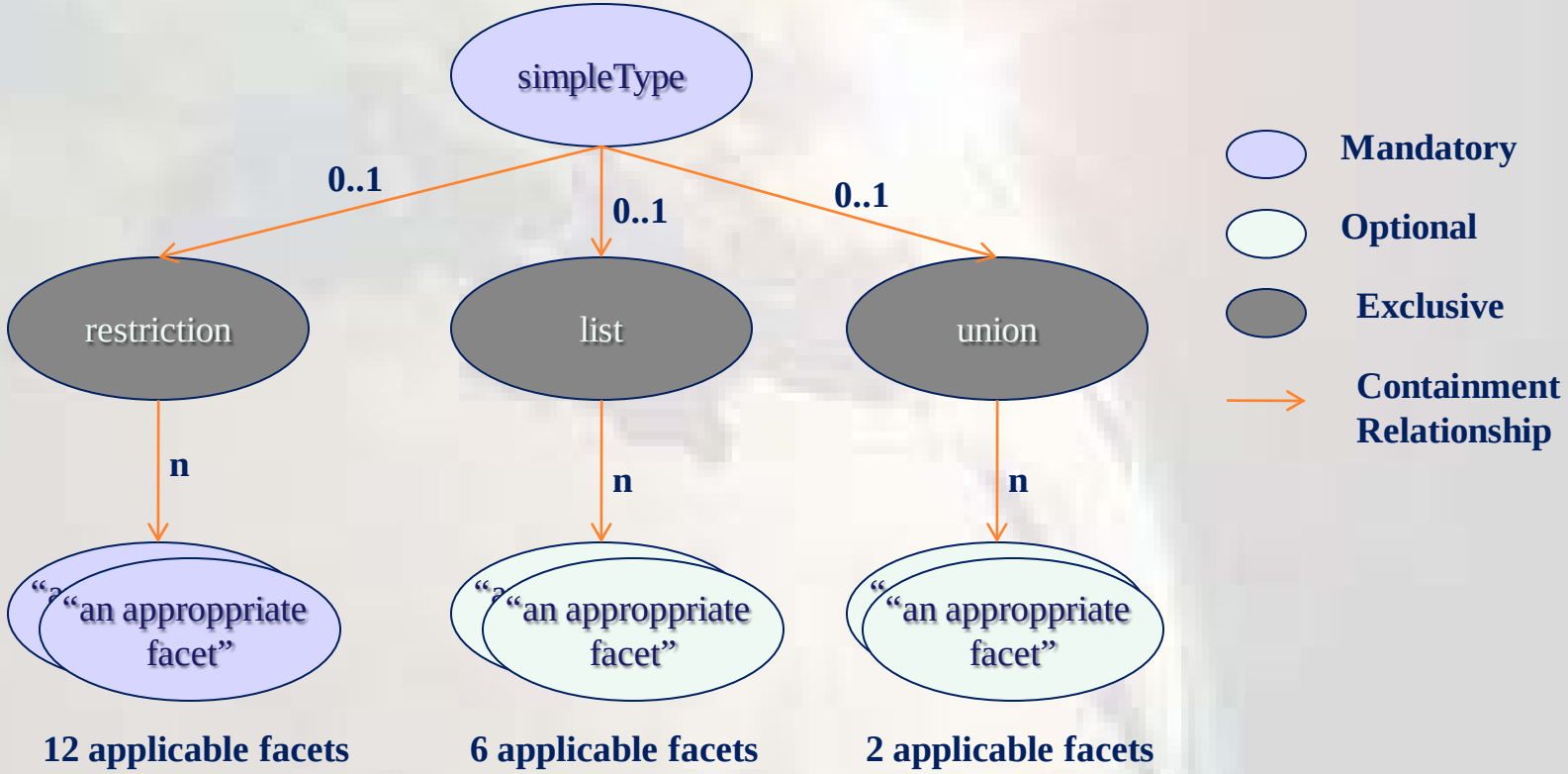
```
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
.. .. </xsd:schema>
```

# Some of subordinate element types of schema element.

- `element`: declares an element used in an XML instance.
- `attribute`: declares an attribute used in an XML instance.
- `simpleType`: this element defines a simple type, which is an XML schema *built-in* type.
- `complexType`: this definition typically contains XML elements and carry attributes, all declared within the type definition.

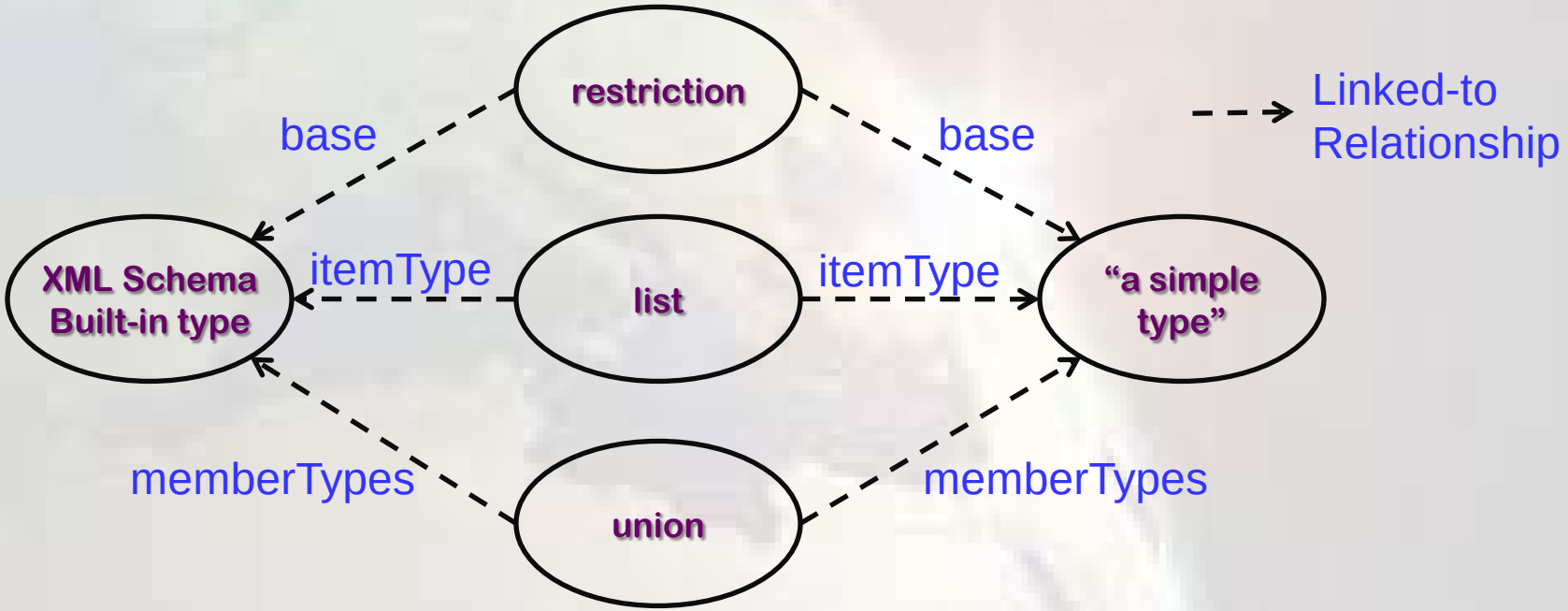
# XML schema containment structure of a simple type definition

- Only one exclusive element must be contained in the superior element.




Source, [2]

# Attribute links between simple type XML schema elements




**Source, [2]**

# A type definition for the title element

```
<xsd: simpleType name="titleTypeUK">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Miss"/>
      <xsd:enumeration value="Mr"/>
      <xsd:enumeration value="Mrs"/>
      <xsd:enumeration value="Ms"/>
   </xsd:restriction>
</xsd:simpleType>
```

Global elements may appear at the top level of an XML instance:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      . . .
      <!-- "titleTypeUK" definition goes here -->
      <xsd:element name="title" type="tns:titleTypeUK"/>
</xsd:schema>
```

# A type definition for the first-name element

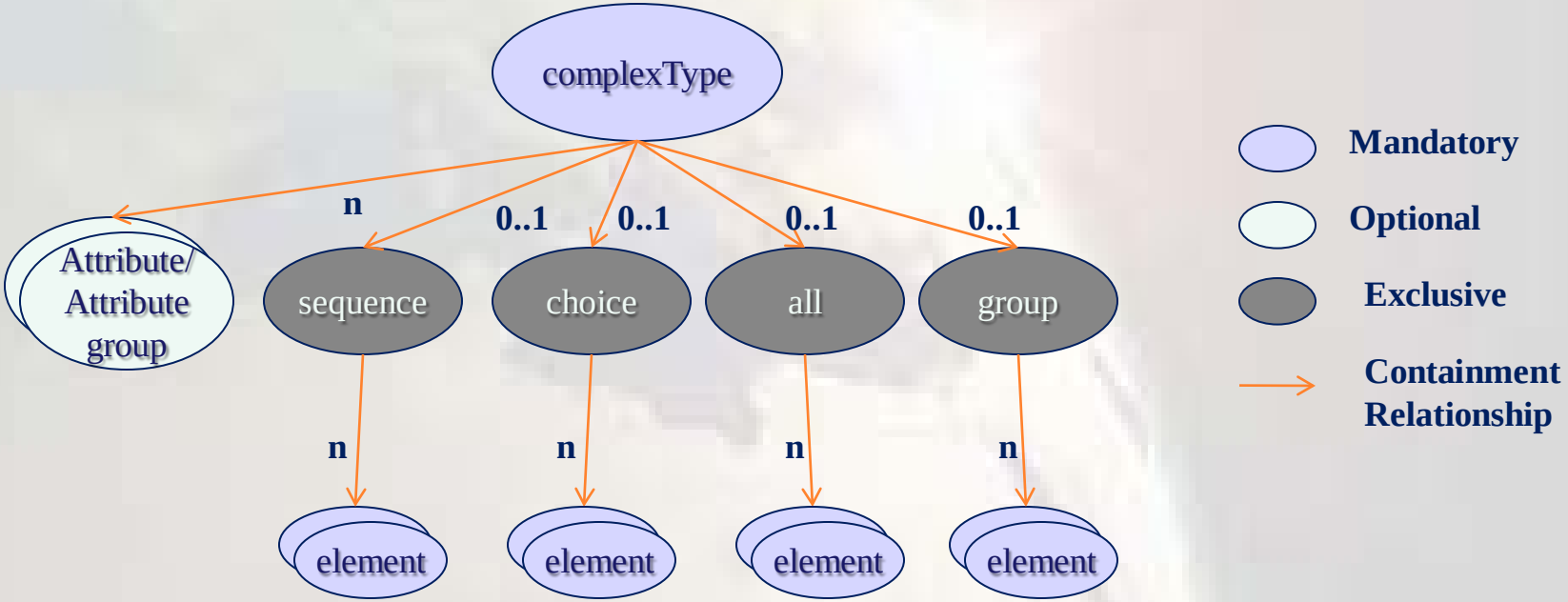
```
<xsd: simpleType name="firstNameType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z][a-z]*"/>
   </xsd:restriction>
</xsd:simpleType>
```

## minLength and maxLength

```
<xsd: simpleType name="firstNameType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z][a-z]*"/>
      <xsd:minLength value="2"/>
      <xsd:maxLength value="20"/>
   </xsd:restriction>
</xsd:simpleType>
```

# XML schema containment structure of a complex type definition

- Only one exclusive element must be contained in the superior element. And an arbitrary number of attribute.




Source, [2]

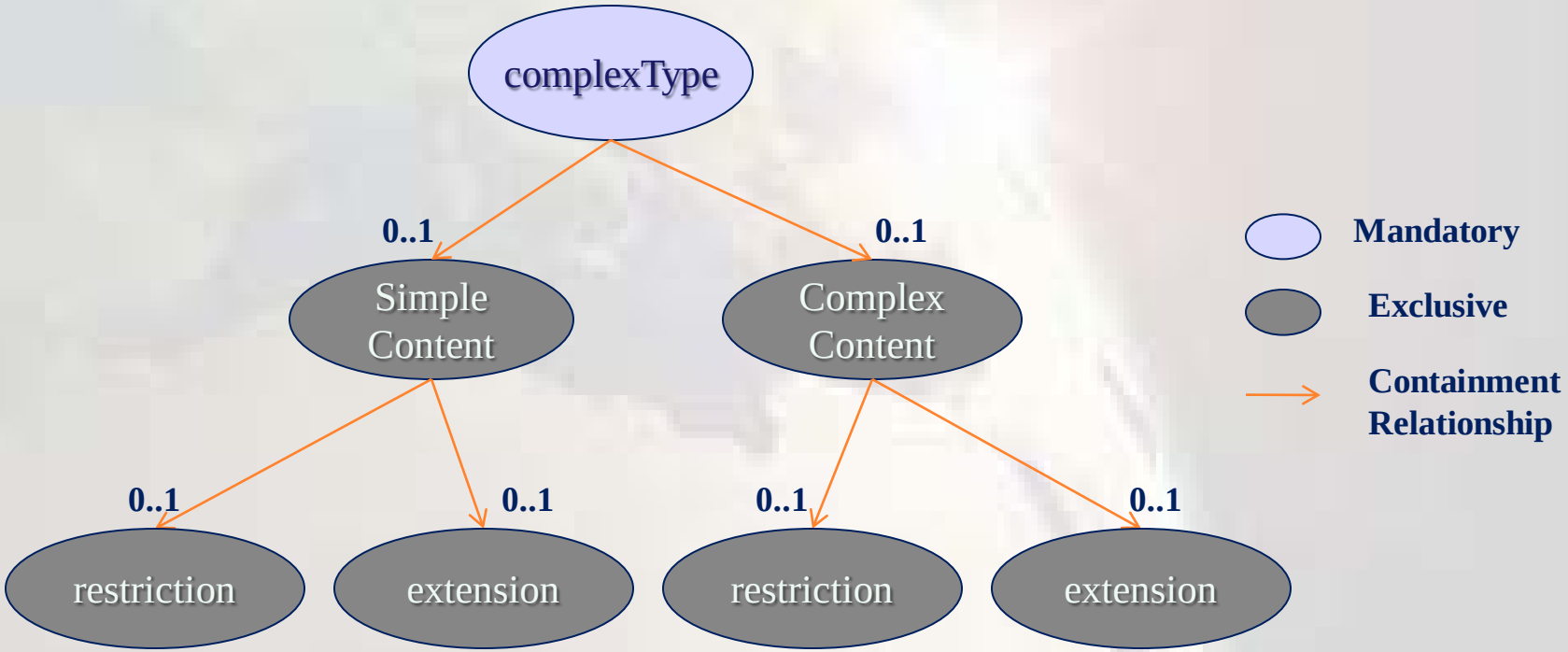
# Complex type definitions

- The following XML schema presents a group named `nameGroup` that is referred to within a complex type definition, [2].

```
<!-- this is the named group definition.
<xsd:group name="nameGroup">
  <xsd:sequence>
    <!-- Here is the type definition of a name -->
  </xsd:sequence>
</xsd:group>
<!-- Here the named group if referred to within a complex type. -->
<xsd:complexType name="addressType">
 <xsd:sequence>
    <!-- Here is the reference to the above defined group -->
    <xsd:group ref="nameGroup"/>
       . . .
  </xsd:sequence>
</xsd:complexType>
```

# XML schema containment structure for deriving types by extension

- `simpleContent` and `complexContent` elements must be superior to either restriction or extension elements.




**Source, [2]**

# Linking XML schemas to XML instances

- `schemaLocation` attribute contains value pairs:
  - The first value is a namespace.
  - The second, provides a link to the schema used for validating elements and attributes contained in this namespace.

# Linking schemas to instances

- To link the address document instance to the address schema via the `schemaLocation` attribute, we need to add this attribute to the instance document.

Example

```
<cX:address
  xmlns:cX="http://companyx.com/ns/employees"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://companyx.com/ns/employees
                      http://localhost/address.xsd">
      . . .
   <!-- the first URI defines the namespace for elements and   -->
   <!-- attributes in this document, for which the second URI  -->
   <!-- points to the schema location of the schema that       -->
   <!-- can be used to validate these elements and attributes  -->
</cX:address>
```

# Summary

During this session we have explained with examples the following:

1. XML namespaces
2. XML schema

Next session will cover Xpath and Xquery.

# References


1. Bray T, Paoli J, Sperberg-McQueen C M, Maler E. Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, http://www.w3.org/TR/REC-xml, 2000.
2. Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, “Perspectives on Web services-Applying SOAP, WSDL and UDDI to real-world projects, 2nd edition, Springer, 2005.

# Thanks

*Mohammed Aldasht*