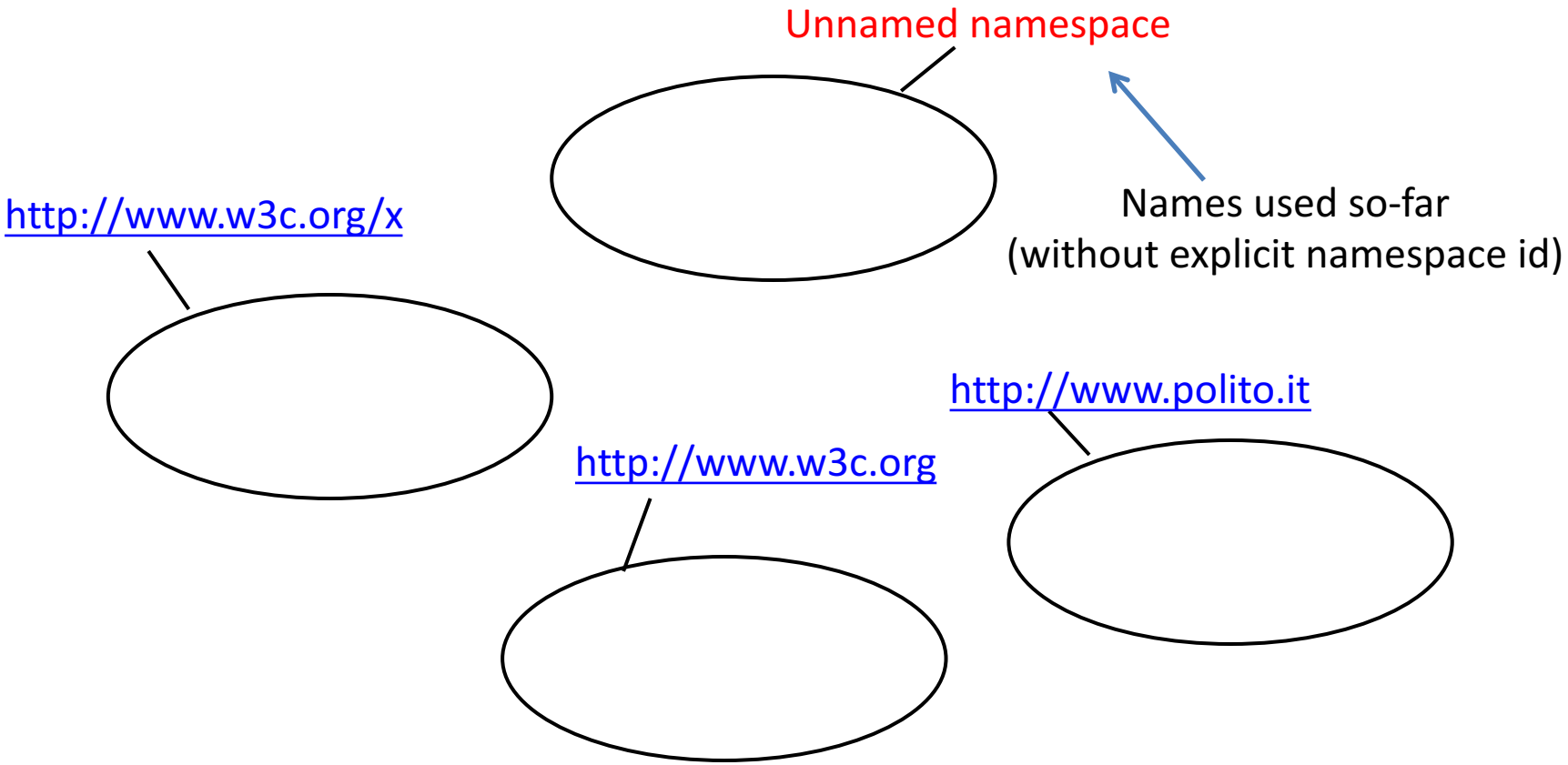
XML Namespaces and XML Schema

XML Namespaces

- A technique to avoid *name clashes* on XML elements and attributes
- useful especially when *uniqueness* of element or attribute names must be guaranteed (e.g. for standards)
- The principle is the creation of disjoint name spaces, identified by URL-like hierarchical identifiers
  - The same name can be re-used in different name spaces with different meanings

Namespace Organization

- Each namespace is a set of names uniquely identified by an *IRI* (extended form of URI)



Qualified Names

- Direct use of IRIs is cumbersome,

  => a short symbolic local identifier is assigned to each namespace used within a document

- A *qualified name* is a concise representation of the pair that uniquely identifies a name with its namespace
- A qualified name may take one of two forms:
  - A name with a prefix: the name belongs to the namespace locally identified by the prefix
  - A name without prefix: the name belongs to the *default namespace* (the unnamed namespace if no default applies)
- Examples of qualified names:

```
ns1:letter      ns2:letter          letter
```

Namespace Declaration

- In order to refer a namespace, a *namespace declaration* is needed in the initial tag of an enclosing element
  - The declaration assigns a *local symbolic identifier* to the namespace
  - or defines it as the *default namespace* (for element names only)

Namespace Declaration

- A namespace is declared by a special attribute
  - whose **name** is
    - `xmlns` (declaration of the default namespace)

    or
    - `xmlns:` followed by the local symbolic name of the namespace
  - whose **value** is the IRI that identifies the namespace
- The scope of a namespace declaration is the whole element in which it occurs
- Examples:

```
<section xmlns="http://www.w3c.org"> … </section>
<section xmlns:ns1="http://www.x.y"
         xmlns:ns2="http://www.x.z"> … </section>
```

An Example of Namespace Use

```
<?xml version = "1.0"?>

<directory xmlns = "http://www.polito.it/xml/plain"
           xmlns:image = "http://www.polito.it/xml/image">

      <file filename="book.xml">
            <description>A book list</description>
      </file>

      <image:file filename="funny.jpg">
            <image:description>A funny picture
            </image:description>
            <image:size width="200" height="100"/>
      </image:file>

</directory>
```

XML Schema

- Schema: XML application for describing XML applications
- Schemas are more powerful than DTDs :
  - A schema can describe more precisely the constraints that XML documents must adhere to
  - Since a schema is itself an XML document
    $\Rightarrow$ It can be validated and manipulated using standard XML techniques
- Different schema languages exist and more can be defined
- In these slides we refer to W3C XML Schema (1.0, 2nd edition). XML1.1 is a slight extension.

W3C Schema (XML Schema)

- Reference Namespace:

  http://www.w3.org/2001/XMLSchema

- Reference DTD (non-normative):

  "-//W3C//DTD XMLSCHEMA 200102//EN"

  "http://www.w3.org/2001/XMLSchema.dtd“

- Reference schema (auto-description)

  http://www.w3.org/2001/XMLSchema.xsd

- Tutorial (reference for study):
  - XML Schema Part 0: Primer (W3C Recommendation)

The Structure of a Schema

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

    Annotations

    Global Element and Attribute declarations

    Type (data/models) and Group definitions

</xsd:schema>
```

Example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
      <xsd:documentation> Purchase order schema for Example.com.
                          Copyright 2000 Example.com.
                          All rights reserved.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
    <xsd:element name="comment" type="xsd:string"/>
    <xsd:complexType name="PurchaseOrderType">
       <xsd:sequence>
          <xsd:element name="shipTo" type="USAddress"/>
          <xsd:element name="billTo" type="USAddress"/>
          <xsd:element ref="comment" minOccurs="0"/>
          <xsd:element name="items" type="Items"/>
       </xsd:sequence>
       <xsd:attribute name="orderDate" type="xsd:date"/>
    </xsd:complexType>
...
</xsd:schema>
```

← annotation

← global element declarations

← type definition

Types

- Each type
  - describes a set of **valid strings**
  - can be identified by a **name**
- **SIMPLE TYPES :**
  - Describe *strings without markup*

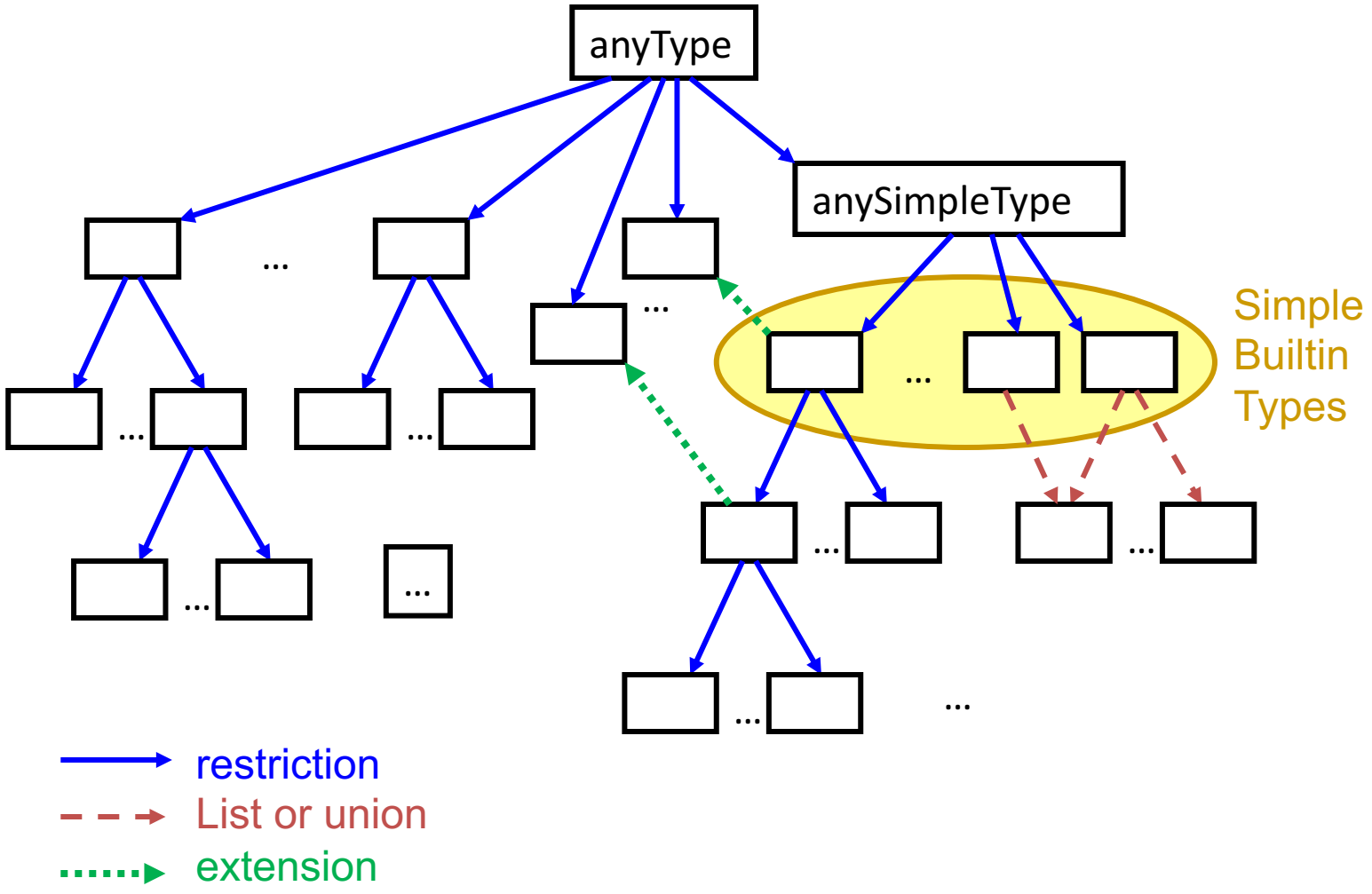
  ⇒ Unstructured data (e.g. the possible values of an attribute)

  (e.g. the contents of an element with no nested elements)
- **COMPLEX TYPES :**
  - Describe *elements*
    - Admitted attributes
    - Admitted Content

  ⇒ Structured data (the data associated with an element)

  ⇒ An element with no attributes and no nested elements is said to have simple type

Type System Organization



Type System Organization

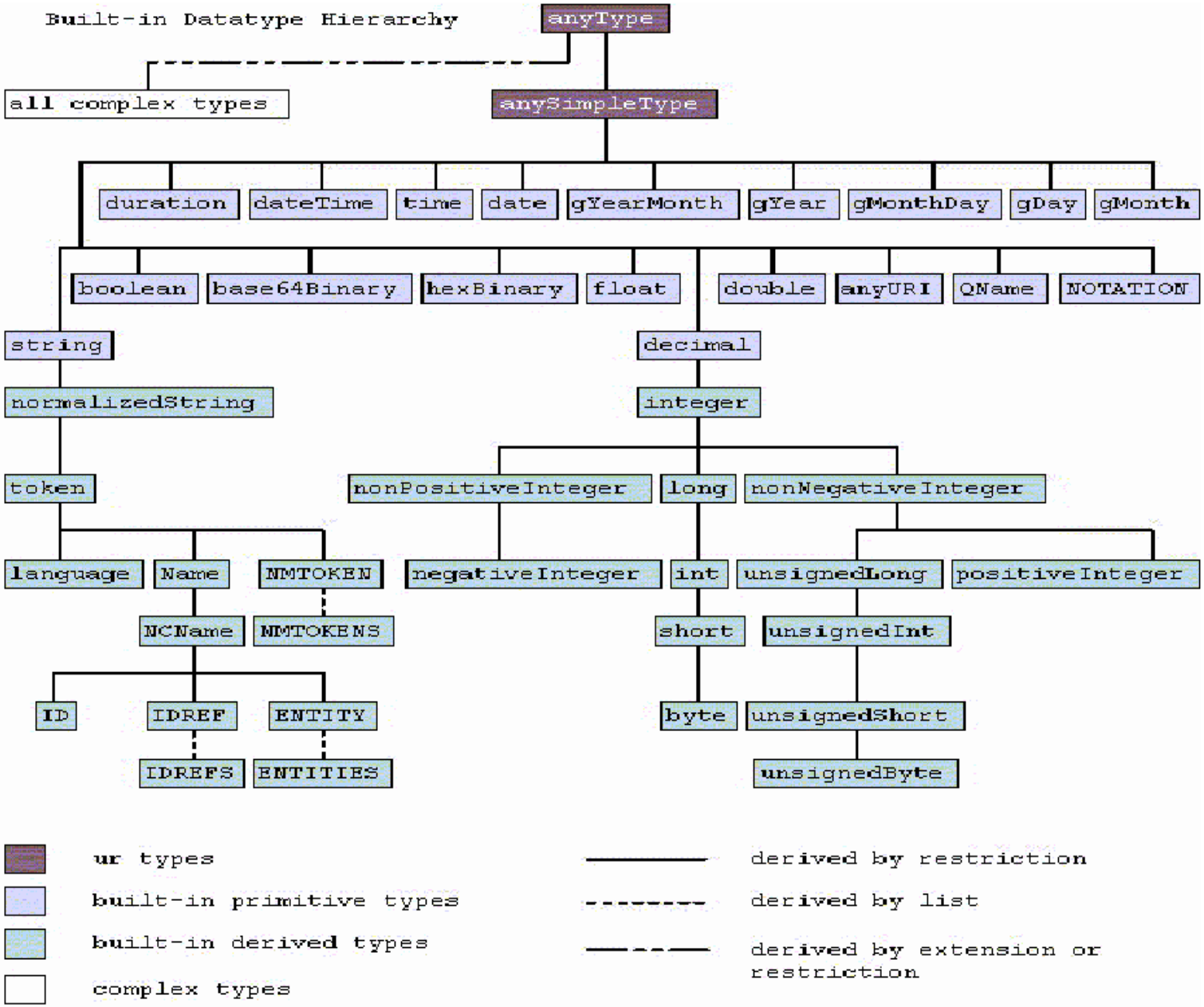
- Types form a **hierarchy**
- The root of the hierarchy is the **`anyType`** type
- The sub-hierarchy of simple types is rooted at **`anySimpleType`**
- Each **simple** type can be:
  - built-in (predefined)

  or
  - derived by restriction or by list from another simple type
- Each **complex** type (except **`anyType`**) is
  - a restriction of a complex type

  or
  - an extension of a simple type or of a complex type

Simple Type Definitions

- Simple types can be:
  - Built-in (predefined)
  - Defined explicitly (derived) by:
    - Restriction (definition of a sub-type) of simple types
    - Construction
      - List types
      - Union types
- Explicit definition is done by a **`SimpleType`** element containing a **`restriction`**, **`list`** or **`union`** element


Built-in Datatype Hierarchy
anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
complex types
derived by restriction
derived by list
derived by extension or restriction

Simple Types Defined by Restriction

- Their values are a subset of the ones of a base simple type
- Syntax:

```
<xsd:simpleType name="…">
    <xsd:restriction base="…">
     …
    </xsd:restriction>
</xsd:simpleType>
```

Base type to be restricted

Facet

- Example:

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
```

List Types

- Simple types whose values are sequences of simple data, **all of the same type**.
  - Predefined list types: NMTOKENS, IDREFS, ENTITIES.
- Syntax:

```
<xsd:simpleType name="…">
  <xsd:list itemType="…"/>
</xsd:simpleType>
```

List base type

- Example:

```
<xsd:simpleType name="ListOfMyIntType">
  <xsd:list itemType="myInteger"/>
</xsd:simpleType>
```

Union Types

- Simple types whose values are sequences of simple types. The type of each list item belongs to a given set of types.
- Syntax:

```
<xsd:simpleType name="…">
  <xsd:union memberTypes="…"/>
</xsd:simpleType>
```

Union base types

- Example:

```
<xsd:simpleType name="IntegersAndDates">
  <xsd:union memberTypes="xsd:integer xsd:date"/>
</xsd:simpleType>
```

Restrictions of List and Union Types

- List and Union types have their own restriction facets
- Example:

```
<xsd:simpleType name="USStateList">
 <xsd:list itemType="USState"/>
</xsd:simpleType>

<xsd:simpleType name="SixUSStates">
 <xsd:restriction base="USStateList">
  <xsd:length value="6"/>
 </xsd:restriction>
</xsd:simpleType>
```

The main Facets

- length
- minLength, maxLength
- pattern
- enumeration
- maxExclusive, maxInclusive, minExclusive, minInclusive
- totalDigits, fractionDigits

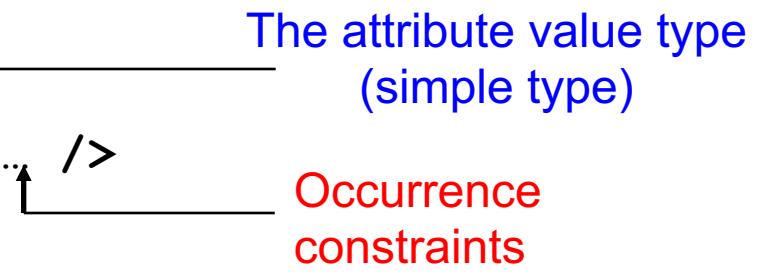
Definition of Complex Types

- A complex type is defined by a **`complexType`** element that includes the element model description.
- The model can be expressed:
  1. By direct construction:
     - Declaration of allowed attributes
     - Description of the content model (includes declaration of allowed sub-elements)
  2. By derivation:
     - Restriction of complex types
     - Extension of simple or complex types

Attribute Declarations

- Syntax:

```
<xsd:attribute name="…" type="…" … />
```



- Occurrence constraints are expressed by the attributes
  - **`use`** specifies if attribute is : `required`, `optional`
  - **`fixed`** means attribute has fixed value (specified by the value of this attribute)
  - **`default`** specifies the default value (the value taken if attribute is absent)

Examples

<xsd:attribute name= "code" type=“xsd:int" use="required"/>

<xsd:attribute name= "country" type=“xsd:string" fixed=“Italy"/>

<xsd:attribute name= “length" type=“xsd:int" default=“0"/>

Content Model

- specifies **order**, **type**, **optionality** and **frequency** of sub-elements or sub-models
- is described by one of the following elements
  - `xsd:sequence` ordered sequence of the specified *sub-elements* or *sub-models*
  - `xsd:choice` one at choice of the specified *sub-elements* or *sub-models*
  - `xsd:all` all of the specified *sub-elements*, in any order (only top-level)
- the possibility of occurrence of an element is specified by an *element declaration* (an `xsd:element` **element**)
  - The type associated with an element is specified by the **`type`** or **`ref`** attribute or by a *nested type definition*

Content Model

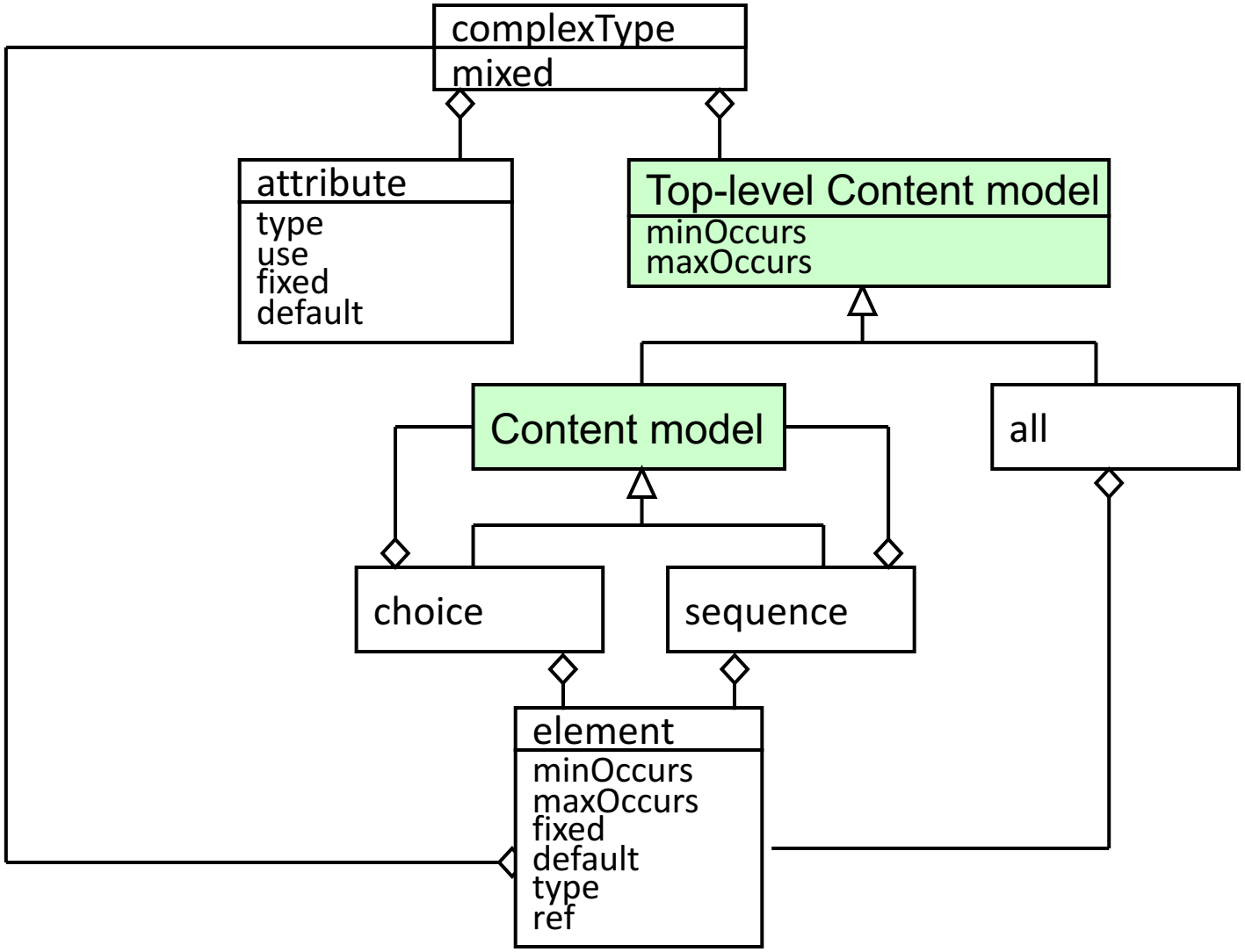
- Frequency and optionality are specified by the attributes:
  - `minOccurs` minimum number of occurrences (default: 1)
  - `maxOccurs` maximum number of occurrences (default: 1)
  - `default` default content (if the element is empty this value is returned by the processor)
  - `fixed` fixed content (the element must have this content)

Examples:

```
<xsd:element name="choice" type="choiceKind" maxOccurs=
  "unbounded" />

<xsd:element name= "field" type="fieldtype" default= "empty
  field">          …
<xsd:element/>
```

Complex Types (Direct Definition)



Example

```
<xsd:complexType name="PurchaseOrderType">
 <xsd:sequence>
  <xsd:element name="singleUSAddress">
    <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="name" type="xsd:string"/>
           <xsd:element name="address" type="xsd:string"/>
           <xsd:element name="zip" type="xsd:decimal"/>
        </xsd:sequence>
        <xsd:attribute name="country" type="xsd:NMTOKEN"
         fixed="US"/>
     </xsd:complexType>
  </xsd:element>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="items" type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Special Models

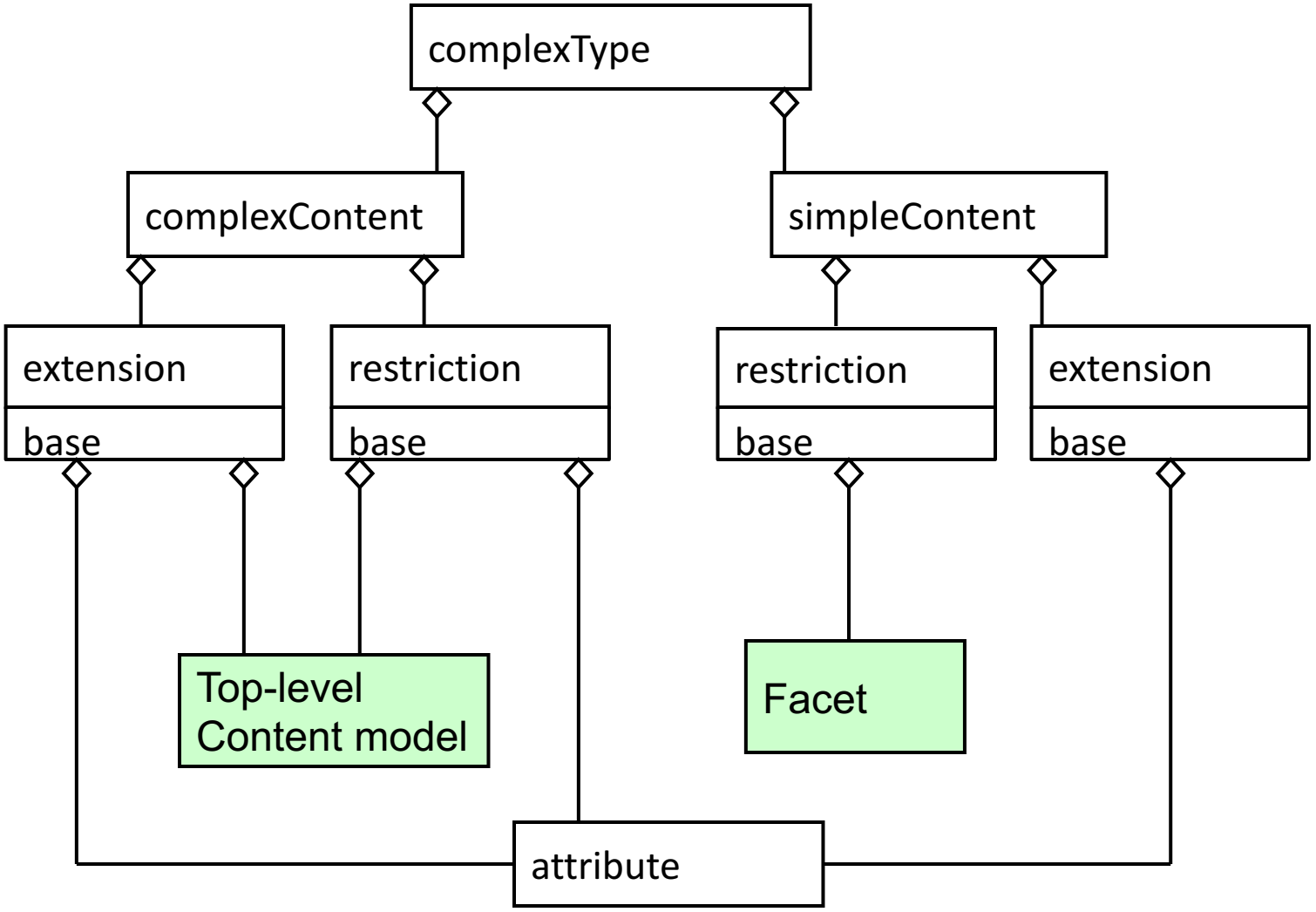
- Mixed content models
  - The attribute **`mixed="true"`** in **`xsd:complexType`** specifies that text can occur between any two sub-models (without restrictions).
- Empty model (content is absent)
  - Is simply specified omitting the model description inside the **`xsd:complexType`** element.
  - **Example**:

```
<xsd:complexType name="internationalPrice">
  <xsd:attribute name="currency" type="xsd:string">
  <xsd:attribute name="value" type="xsd:decimal">
</xsd:complexType>
```

Derived Complex Models

- A complex type can be derived
  - by extension
    - of a simple type: attributes are added to a simple type model
    - of a complex type: new elements can be added (appended by sequence) to the ones of the base type
  - by restriction
    - The possible contents are restricted by facets (the model is redefined)
- Derivation is specified introducing a **`complexContent`** or **`simpleContent`** element (according to the type of content model) inside a **`complexType`**

Derived Complex Types



Example: Extension of a simple type

```
<internationalPrice currency="EUR">423.46</internationalPrice>
```

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
   <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="currency" type="xsd:string"/>
    </xsd:extension>
   </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Example: Extension of a complex type

```
<xsd:complexType name="Address">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="city" type="xsd:string"/>
 </xsd:sequence>
</complexType>

<xsd:complexType name="USAddress">
 <xsd_complexContent>
  <xsd:extension base="Address">
   <xsd:sequence> <xsd:element name="state" type="USState"/>
    <xsd:element name="zip" type="xsd:positiveInteger"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Example:
Equivalent type defined from scratch

```
<xsd:complexType name="USAddress">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="city" type="xsd:string"/>
  <xsd:element name="state" type="USState"/>
  <xsd:element name="zip" type="xsd:positiveInteger"/>
 </xsd:sequence>
</xsd:complexType>
```

Example: Restriction of a complex type

```
<xsd:complexType name="RestrictedPurchaseOrderType">
  <xsd:complexContent>
    <xsd:restriction base="PurchaseOrderType">
      <xsd:sequence>
        <xsd:element name="singleUSAddress"/>
          <xsd:complexType>
            ...
          <xsd:complexType>
        <xsd:element name="billTo" type="USAddress"/>
        <xsd:element ref="comment" minOccurs="1"/>
        <xsd:element name="items" type="Items"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

Need to repeat whole sequence

Was "0"

No need to add attribute (inherited)

Example of a Complete Schema (The Purchase Order)

- Schema file: PurchaseOrder.xsd
- XML document file: PurchaseOrder.xml

Groups

- The **`group`** element enables the definition of a content model that can be referenced by name
- The **`attributeGroup`** element enables a set of attributes to be packed inside a named group (that can be later referenced by name)

Example

```
<xsd:complexType name="PurchaseOrderType">
 <xsd:sequence>
  <xsd:choice>
   <xsd:group ref="shipAndBill"/>
   <xsd:element name="singleUSAddress" type="USAddress"/>
  </xsd:choice>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="items" type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<xsd:group name="shipAndBill">
 <xsd:sequence>
  <xsd:element name="shipTo" type="USAddress"/>
  <xsd:element name="billTo" type="USAddress"/>
 </xsd:sequence>
</xsd:group>
```

Nillable Elements

- The absence of the information included in an element can be signaled by omitting the element
- In addition, XML-Schema enables elements with explicit indication of being “nil”
- Example:
  - element declaration in the schema

```
<xsd:element name="shipDate" type="xsd:date" nillable="true"/>
```

  - nil element instance in the document

```
<shipDate xsi:nil="true“
xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance”></shipDate>
```

Unique Identifiers and References

- XML schema simple built-in types include ID, IDREF and IDREFS
  - by these, the same controls available in DTDs are possible
- In addition, more flexible and powerful mechanisms for specifying unique identifiers and references are available

  => The scope of uniqueness/references can be specified

  => Identifiers and references can be
  - a single attribute
  - a single element
  - combinations of data (e.g. the combination of two attributes)

Specifying Uniqueness

- The requirement of having elements that are uniquely identified can be specified by a `unique` element
- The `unique` element includes:
  - one `selector` element specifying the **scope** (what elements are uniquely identified)
  - `field` elements specifying the **identifiers**: combination of fields that uniquely identify each element specified by the selector
- Selector and field elements use xPath expressions to specify sets of elements and fields
  - only a subset of xPath is accepted
  - expressions must be relative to the position of the unique element, which must be put inside an element declaration

Example

```
<xsd:element name="persons">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="person" type="Person"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:unique name="unique_persons">
    <xsd:selector xpath="person">
    <xsd:field xpath="@name"/>
  <xsd:unique/>
</xsd:element>

<xsd:complexType name="Person">
  <xsd:sequence>
    …
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

Specifying Keys and References

- Keys are unique identifiers whose fields are also mandatory
  - Keys are specified by `key` elements (similar to `unique`)
- The fact that some fields can be used as references to unique identifiers is specified by a `keyref` element that
  - refers to a `key` (or `unique`) element by name (attribute `refer`) to specify the identifiers to which the reference points
  - has `selector` and `field` sub-elements that specify
    - the elements where the reference is included
    - the field(s) used as references (order and type must match)
- `key` and `keyref` elements must be in the same element or in elements bound by ancestor relationship

Example: The Quarterly Report

- A report for purchase orders:
  - XML document example: **4Q99_nn.xml**
  - XML schema: **report_nn.xsd**

- Uniqueness/reference constraints:
  - the regions list must not include two zip elements with the same code (uniqueness constraint => use `unique` element)
  - the number attribute of each part element in the parts list must have unique value and must be referenceable, i.e. treated as a primary key (=> use `key` element)
  - the number attribute of each part element in the regions element must equal the number attribute of one part element in the parts list (=> use `keyref` element)

Target Namespace

- The targetNamespace attribute declares the namespace to which the types, elements, and attributes defined in the schema at global level (under the root) will belong
- With no targetNamespace declaration, root elements, attributes and types have no associated namespace (are in the unnamed namespace)
- Example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:po="http://www.example.com/PO"
 targetNamespace="http://www.example.com/PO">
  <element name="purchaseOrder" type="po:PurchaseOrderType"/>
  <complexType name="PurchaseOrderType">
  ...
```

Document-Schema Association

- An xml document can reference one or more schema to be used for its validation:
  - Referencing a schema without target namespace:

```
<?xml version="1.0"?>
<documentroot
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="myschema.xsd">
```

  - Referencing a schema with target namespace:

```
<?xml version="1.0"?>
<documentroot
   xmlns:mns="http://www.mynamespace.it"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.mynamespace.it myschema.xsd">
```

Splitting Schema on Multiple Files

- Splitting can be done by exploiting the `include` and `redefine` elements
- Example:

```
<xsd:include
   schemaLocation="http://www.example.com/schemas/address.xsd"/>
```

- All the global declarations and type definitions found in the referenced file are included into the including file (the target namespace of the including and included file must be the same).
- It is the responsibility of the processor to resolve included schema files using the mandatory **schemaLocation** attribute
- Redefine is similar, but types can be modified when included

Reusing declarations belonging to other namespaces

- A schema can re-use declarations appearing in another schema with a different target namespace using the `import` element
- Example:

```
<xsd:import namespace="http://www.example.com/IPO"/>
```

- Optionally, the import element can have a **schemaLocation** attribute that specifies where the schema for that namespace can be found

Examples

- The International Purchase Order (with targetnamespace and include)
  - XML document example: **ipo.xml**
  - XML schema: **ipo.xsd**
  - XML schema for international addresses: **address.xsd**
- Quarterly report (with targetnamespace and import)
  - XML document example: **4Q99.xml**
  - XML schema: **report.xsd**
  - Imported XML schema: **ipo.xsd**

Tool Support

- XML-schema editors
- Validators
- Language binding tools (e.g. JAXB)
  - Generate code for transparent read/write access to XML documents with validation