INFO-H-509 : Technologies XML
TP 2 - Schema languages for XML

Professeur : Stijn Vansummeren
Assistant : Michaël Waumans
http://cs.ulb.ac.be/public/teaching/infoh509

XML Schema Validity

All documents required for completing the exercises below are available on the course's web page.

Exercise 1.1

Propose corrections of the XML document so that this one becomes valid with respect to the schema.

- The element `description` must appear first in the sequence.
- The model `all` imposes that all the elements do appear. In particular, the element `accessibility` is required for the element `root` ua4.218.
- The element `phone` is not authorized as a child of the element `room`. It must be deleted.
- The capacity of a room must be at least of 10 persons. The capacity of the room UB4.329a must then be corrected.

Exercise 1.2

Proceed according the the following steps to complete the schema :

- Complete the definition of `eventtype` to take into account the elements common to the courses and practical sessions.

  The attributes `on`, `id`, as well as the element `room` are common to the exercices and theoretical courses. So, we modify the schema in the following manner.

```
<xs:complexType name="eventType" abstract="true">
    <xs:sequence>
        <xs:element name="room" type="xs:string" />
    </xs:sequence>
    <xs:attribute ref="id" use="optional" />
    <xs:attribute name="on" type="xs:dateTime" use="required" />
</xs:complexType>
```

  We define this type as abstract, so that it cannot be used in a document instance.

- Extend this type separatly for the courses and practical sessions.

  In both cases, it will be necessary to add the elements `topic`. The first difference is about the content, and thus, the type of those. For a course, `topic` possess an attribute `name` and contains different elements `resource`. For a practical session, `topic` will simply have the type `string`. The following source code creates the appropriate type for the element `topic` of a course.

```
<xs:complexType name="lectureTopic">
    <xs:sequence>
        <xs:element name="resource"
            minOccurs="0"
            maxOccurs="unbounded"
            type="xs:anyURI"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
</xs:complexType>
```

On this basis, we extend the type `eventType`. Remark the use of the indications about cardinalities to satisfy that a practical session concerns at most three courses.

```
<xs:complexType name="lectureType">
    <xs:complexContent>
        <xs:extension base="eventType">
            <xs:sequence>
                <xs:element name="topic"
                            maxOccurs="unbounded"
                            type="lectureTopic" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="exercisesType">
    <xs:complexContent>
        <xs:extension base="eventType">
            <xs:sequence>
                <xs:element name="topic"
                            minOccurs="1"
                            maxOccurs="3"
                            type="xs:string" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

- Complete the definition of the element `course`. Notice that the courses and practical sessions can be interlaced.

  The idea is to authorize the repetition of a choice between `lecture` and `exercices` :

```
<xs:complexType name="courseType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="lecture" type="lectureType" />
        <xs:element name="exercises" type="exercisesType" />
    </xs:choice>
    <xs:attribute ref="id" use="required" />
</xs:complexType>

...
    <xs:element name="course" type="courseType" />
...
```

Exercise 1.3

Additional exercise : Use the elements `unique`, `key` and `keyref` to garantee that the ressources of a course are unique, that a course refers to an existing room and that a practical session has for subject an existing course. Correct the document if necessary, to satisfy those constraints.

The unicity constraint is garanteed by changing the element `lecture` as follows :

```
<xs:element name="lecture" type="lectureType">
    <xs:unique name="resUnique">
        <xs:selector xpath="topic/resource" />
        <xs:field xpath="." />
    </xs:unique>
</xs:element>
```

Remark that `selector` contains all the elements for which we want to garantee a constraint. `field` defines a field for each element on which the constraint must be verified. `field` must then be unique for each element of the group `selector`.

The foreign key constraint of an event to a local is garanteed this way :

```
<xs:element name="schedule">
    ...

    <xs:key name="roomkey">
        <xs:selector xpath="room" />
        <xs:field xpath="@id" />
    </xs:key>
    <xs:keyref name="eventroom" refer="roomkey">
        <xs:selector xpath="course/exercises | lecture" />
        <xs:field xpath="room" />
    </xs:keyref>
</xs:element>
```

The constraint between a practical session and a theoretical course cannot be satisfied in the same way : the element key requires that all the fields must be defined. Since the attribute `id` is optional for a course, we must use `unique`.

```
<xs:element name="course" type="courseType">
    <xs:unique name="lectureid">
        <xs:selector xpath="lecture" />
        <xs:field xpath="@id" />
    </xs:unique>
    <xs:keyref name="exercisestopic" refer="lectureid">
        <xs:selector xpath="exercises/topic" />
        <xs:field xpath="." />
    </xs:keyref>
</xs:element>
```

Exercise 1.4

As indicated by the reference book pp 139–140, the value of the attribute `elementFormDefault` must be `qualified` to use the usual semantic of namespaces. Also remark that the type of the element message know uses the namespace.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:m="http://www.example.org/message"
    targetNamespace="http://www.example.org/message"
    elementFormDefault="qualified"
    >
    <xs:complexType name="message">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="on" type="xs:dateTime" use="required" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

    <xs:element name="message" type="m:message" />
</xs:schema>
```

Regular expressions and DTD

Exercise 2.1

	$a(bb?)?(c^+(a\|d))$	a^+bbccd	$c?b?(a^*\|b\|c)^+$	$c^*(abcabc)$	$(a^+b)^+(cd)^+$
abcabc			O	O	
abcd	O				O
abbccd	O	O			
cabcabc			O	O	
aaccd					
abaaabcd					O
Déterministe ?	O	O		O	O

To show that the third expression is non deterministic, consider b_1a1 et b_2a_1.

Exercise 2.2

Complete the DTD corresponding to the first exercise : apply similar constraints to the element `course` if possible.

Let's begin by adding the missing definitions :

```
<!ELEMENT course (lecture|exercises)*>
<!ELEMENT lecture (room, topic*)>
<!ELEMENT exercises (room, topic*)>
<!ELEMENT topic (resource)*>
<!ELEMENT resource (#PCDATA)>

<!ATTLIST lecture id CDATA #IMPLIED
                  on CDATA #REQUIRED>
<!ATTLIST exercises id CDATA #IMPLIED
                    on CDATA #REQUIRED>
<!ATTLIST topic name CDATA #IMPLIED>
```

This does not resolve all the problems. For example, the elements `room` and `topic` contain, depending on their position in the document, either text or diverse childs. DTD does not allow to differenciate the context. We do have to either change the definition of the XML document, or accept at the same time the two forms of content. In the last case, the document will no always be valid as wished. However, it is not always practical to change the source documents.

Here, `room` and `topic` are modified to accept both syntaxes. To do so, we use the *Mixed definitions* of DTD that accept interlaced text and elements. The definition of the attribute `id` must also be change from required to optional :

```
<!ELEMENT topic (#PCDATA | resource)*>
<!ELEMENT room (#PCDATA | friendlyName | beamer | capacity | accessibility)*>
<!ATTLIST room id CDATA #IMPLIED>
```

Those modifications finally allow the validation of the document to the price of validation rules that are not very restrictive.

Exercise 2.3

Compare both schema, DTD and XML Schema. What are the main differences ?

DTD does not offer equivalent expression to `all` from the XML Schema. To express the same constraint, we have to enumerate explicitly all of the possible cases. For example, for the elements a, b and c :

$$(a,(b,c)|(c,b))|(b,(a,c)|(c,a))|(c,(a,b)|(b,a))$$

DTD associates the content of an element to the name of this one. In the contrary, XML Schema associates the content of an element to the type of this one .

DTD does not allow to put a constraint one text data. XSD do offer numerous base types (`string`, `boolean`, `integer`, etc.) and allows to constrain those with facets (e.g. constraint of the room capacity).

DTD does not allow to define only one key by document. XSD does allow multiple keys and unicity constraints. XSD also allows the use of composite keys.

The definition of keys in DTD/XSD must be done using the attribute types ID/IDREF. Thus, it is impossible to use the text of an element as a key, as it is the case in XSD.

However, DTD is advantageous because it is quite easy to read and write.