INFO-H-509 : Technologies XML
TP 2 - Langages de schémas pour XML

Professeur : Stijn Vansummeren
Assistant : François Picalausa
http://cs.ulb.ac.be/public/teaching/infoh509

Année académique 2009-2010

XML Schema Validity

Le données de ce TP sont disponibles en téléchargement sur le site des TPs.

Exercice 1.1

Proposer des corrections dans le document XML pour que celui-ci devienne valide en rapport du schéma.

- L'élément `description` doit apparaître en premier dans la séquence.
- Le modèle `all` impose que l'ensemble des éléments apparaisse. En particulier, l'élément `accessibility` est requis pour l'élément `root` ua4.218.
- L'élément `phone` n'est pas autorisé en tant qu'enfant de l'élément `room`. Il doit être supprimé.
- La capacité d'un local doit être d'au moins 10 personnes. Il faut donc corriger la capacité pour le ub4.329a.

Exercice 1.2

Procéder suivant les étapes suivantes pour compléter le schéma :

- Compléter la définition de `eventtype` pour prendre en compte les éléments communs aux cours et aux TPs.

  Les attributs `on`, `id`, de même que l'élément `room` sont communs aux exercices et aux cours théoriques. On modifie donc le schéma de la manière suivante.

```
<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>
```

  On utilise ici `sequence` de manière arbitraire : les éléments doivent être dans un content model `sequence`, `choice` ou `all`. Il faut donc en choisir arbitrairement [1] un. On définit ce type comme abstrait, de sorte qu'il ne puisse pas être utilisé dans un document instance.

1. A noter toutefois que `all` impose que une cardinalité maximale de 1 à son contenu. Ce choix serait donc plus restrictif.

- Étendre ce type séparément pour les cours et pour les TPs.

  Dans les deux cas, if faudra ajouter des éléments `topic`. Une première différence est le contenu, donc le type, de ceux ci. Pour un cours, `topic` possède un attribut `name` et contient différents éléments `resource`. Pour un TP, `topic` aura simplement le type `string`. Le code suivant crée le type approprié pour l'élément `topic` d'un cours.

```
<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>
```

  Sur cette base, nous étendons le type `eventType`. Remarquer l'utilisation des indications de cardinatlité pour satisfaire qu'un TP porte sur au plus trois cours :

```
<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>
```

- Compléter la définition de l'élément `course`. Noter que les cours et séances d'exercices peuvent être entrelacés.

  L'idée est d'autoriser la répétition d'un choix entre `lecture` et `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" />
        ...
```

Exercice 1.3

Exercice supplémentaire : Utiliser les éléments `unique`, `key` et `keyref` pour assurer que les resources d'un cours sont unique, qu'un cours réfère à un local existant et qu'une séance d'exercices a pour sujet un cours existant. Corriger le document si nécessaire, pour satisfaire les contraintes.

La contrainte d'unicité est assurée en changeant l'élément `lecture` comme suit :

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

Remarquer que `selector` contient l'ensemble des éléments pour lesquel on veut assurer une contrainte. `field` définit le champ de chaque élément de l'ensemble sur lequel la contrainte doit être vérifiée. `field` doit donc être unique pour chaque élément de l'ensemble `selector`.

La contrainte de clé étrangère d'un événement vers un local est assurée comme suit :

```
<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>
```

La contrainte entre un TP et un cours théorique ne peut pas être satisfaite de la même manière : l'élément key requiert que l'ensemble des champs soient définis. Puisque l'attribut `id` est optionnel pour un cours, il faut utiliser `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>
```

Exercice 1.4

Comme l'indique le livre de référence pp 139–140, la valeur de l'attribut `elementFormDefault` doit être `qualified` pour utiliser la sémantique usuelle des namespaces. A remarquer aussi, le type de l'élément message utilise maintenant le 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" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

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

Expressions régulières et DTD

Exercice 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

Pour montrer que la troisième expression est non-déterministe, considérer b_1a_1 et b_2a_1.

Exercice 2.2

Compléter la DTD correspondant au premier exercice : appliquer des contraintes similaires à l'élément `course` si possible.

Commençons par ajouter les définitions manquantes :

```
<!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>
```

Ceci ne résoud pas l'ensemble des problèmes. Notamment, les éléments `room` et `topic` contiennent, en fonction de leur position dans le document, soit du texte, soit divers enfants. DTD ne nous permet pas de différencier le contexte. Il nous faut donc soit changer la définition du document XML, soit accepter en même temps les deux formes pour le contenu. Dans ce dernier cas, le document n'aura pas toujours la validité souhaitée. Néanmoins, il n'est pas toujours pratique de changer les documents sources.

Ici, `room` et `topic` sont modifiés pour accepter les deux syntaxes. Pour ce faire, nous utilisons les *Mixed definitions* de DTD qui acceptent l'entrelacement de texte et d'éléments. La définition de l'attribut `id` doit elle aussi être changée, de requise à optionnelle :

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

Ces modifications permettent finalement de valider le document, au prix de règles de validation peu restrictives.

Exercice 2.3

Comparer les deux schémas, DTD et XML Schema. Quelles sont les principales différences ?

DTD n'offre pas d'expression équivalente à `all` de XML Schema. Pour exprimer la même contrainte, il faut énumérer explicitement l'ensemble des cas. Par exemple, pour les éléments a, b et c :

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

DTD associe le contenu d'un élément au nom de celui-ci. Au contraire XML Schema associe le contenu d'un élément au type de celui-ci.

DTD ne permet pas de contraindre les données textuelles. XSD offre de nombreux types de base (`string`, `boolean`, `integer`, etc.) et permet de contraindre ceux-ci avec des facettes (e.g. contrainte de la capacité d'un local).

DTD ne permet de définir qu'une seule clé par document. XSD permet plusieurs clés et contraintes d'unicité. XSD permet aussi l'utilisation de clés composites.

La définition de clés en DTD/XSD doit se faire à l'aide d'attributs de type ID/IDREF. Il est donc impossible de d'utiliser le texte d'un élément comme clé, comme c'est le cas en XSD.

Néanmoins, DTD a pour avantage sa relative simplicité de lecture et d'écriture.