



Séance 12

Les schémas

XML

Définir des règles de validation pour son
encodage

A decorative graphic on the left side of the slide consisting of two overlapping squares. The bottom-left square is a dark blue, and the top-right square is a lighter blue, creating a cross-like shape.

Restreindre le contenu d'élément

Comment préciser ce que peut contenir un
élément dans un encodage

Déclarer du contenu

`<content>` apparaît toujours avant `<attList>` dans l'`<elementSpec>` et il peut contenir :

- Une référence à un élément / classe / module
- `<sequence>` qui définit un enchaînement d'éléments
- `<alternate>` qui définit un enchaînement d'éléments
- `<empty/>` ou `<textNode/>`

```
<elementSpec ident="div" mode="change">  
  <content>  
    <elementRef key="head"/>  
  </content>  
</elementSpec>
```

Dénombrer le contenu

La valeur par défaut de `@minOccurs` et `@maxOccurs` est de 1 et peut avoir pour valeur `zero_ou_plus` | “unbounded”

Pour définir un contenu optionnel, indiquer **`@minOccurs="0"`**

```
<elementSpec ident="div" mode="change">
  <content>
    <elementRef key="head"
      minOccurs="1"
      maxOccurs="1"/>
  </content>
</elementSpec>
```

Définir une séquence de contenu

L'élément `<sequence>` permet de définir une suite précise de contenus : son attribut `@preserveOrder` (true par défaut) permet de spécifier si l'ordre de déclaration est significatif

```
<elementSpec ident="div" mode="change">
  <content>
    <sequence preserveOrder="false">
      <elementRef key="head"/>
      <elementRef key="p"/>
    </sequence>
  </content>
</elementSpec>
```

Imbriquer les séquences

Les séquences peuvent être démultipliées et imbriquées

```
<elementSpec ident="div" mode="change">
  <content>
    <sequence>
      <elementRef key="head"/>
      <elementRef key="p"/>
    </sequence>
    <sequence>
      <alternate>
        <elementRef key="name"/>
        <elementRef key="persName"/>
      </alternate>
      <classRef key="model.pLike"/>
    </sequence>
  </content>
</elementSpec>
```

Définir une alternance de contenu

```
<content>
  <alternate>
    <sequence>
      <elementRef key="street"/>
      <elementRef key="placeName"/>
      <elementRef key="postCode"/>
      <elementRef key="country"
        minOccurs="0"
        maxOccurs="1"/>
    </sequence>
    <elementRef key="addrLine"
      minOccurs="2"
      maxOccurs="4"/>
  </alternate>
</content>
```

Définir un contenu textuel / vide

```
<elementSpec ident="div" mode="change">  
  <content>  
    <sequence>  
      <textNode/>  
      <elementRef key="persName"/>  
    </sequence>  
  </content>  
</elementSpec>
```

```
<elementSpec ident="lb" mode="change">  
  <content>  
    <empty/>  
  </content>  
</elementSpec>
```


Comprendre une séquence de contenu des *Guidelines* : floatingText

```
<content>
  <sequence>
    <!-- peut contenir des éléments de la classe model.global -->
    <classRef key="model.global" minOccurs="0" maxOccurs="unbounded"/>

    <!-- peut contenir une séquence de <front> et des éléments de model.global -->
    <sequence minOccurs="0">
      <elementRef key="front"/>
      <classRef key="model.global" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>

    <!-- DOIT contenir <body> ou <group> -->
    <alternate>
      <elementRef key="body"/>
      <elementRef key="group"/>
    </alternate>

    <!-- peut contenir une séquence de <back> et des éléments de model.global -->
    <sequence minOccurs="0">
      <elementRef key="back"/>
      <classRef key="model.global" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </sequence>
</content>
```

A decorative graphic on the left side of the slide consisting of two overlapping squares. The bottom square is a dark blue, and the top square is a lighter blue, creating a cross-like shape.

Règles de validation

La syntaxe Schematron

Règle schematron et *namespace*

Les règles schematron n'utilisent pas le même *namespace* que le reste de l'ODD qui utilise l'espace de nom TEI : il faut donc déclarer un @xmlns dans l'élément racine et préfixer les règles schematron.

Les chemins renseignés dans la règle schematron doivent être préfixés avec le *namespace* tei.

```
<TEI xmlns="http://www.tei-c.org/ns/1.0"  
      xmlns:s="http://purl.oclc.org/dsdl/schematron">  
  
  [...]   
  <constraint xmlns:tei="http://www.tei-c.org/ns/1.0">  
    <s:assert test="tei:div">[...]</s:assert>  
  </constraint>  
  
</TEI>
```

Contraindre un élément

La règle est contenue dans un élément `<constraintSpec>`. Le langage utilisé est déclaré dans l'attribut `@scheme` et la règle est nommée à l'aide de l'attribut `ident`.

`<s:assert>` permet de définir un test à respecter en [Xpath](#) ainsi qu'un message d'erreur à afficher si la contrainte n'est pas respectée

```
<constraintSpec ident="attReq" scheme="schematron">
  <constraint>
    <s:assert test="@ref or @key">
      L'attribut @ref ou @key doit être renseigné
    </s:assert>
  </constraint>
</constraintSpec>
```

assert / report

`<assert>` renvoie une erreur si la condition dans `@test` est fausse

`<report>` renvoie une erreur si la condition dans `@test` est vraie

Un seul `assert/report` par `constraintSpec`

```
<constraintSpec ident="div" scheme="schematron">
  <constraint>
    <s:assert test="matches(@n, '^[1-9]\d*$')">
      L'attribut n doit être un entier positif.
    </s:assert>
  </constraint>
</constraintSpec>
```

pattern

Pour combiner différentes rule/assert dans la même constraint et définir des variables, etc.

```
<constraintSpec ident="div" scheme="schematron">
  <constraint>
    <s:pattern>
      <s:assert test="@type='chapter'">
        L'attribut type doit être "chapter"
      </s:assert>
      <s:assert test="matches(@n, '^[1-9]\d*$')">
        L'attribut n doit être un entier positif.
      </s:assert>
    </s:pattern>
  </constraint>
</constraintSpec>
```

Ajouter du contexte

`<s:rule>` permet d'ajouter un contexte à l'application à `<s:assert>`

```
<constraintSpec ident="subDiv" scheme="schematron">
  <constraint>
    <s:rule context="tei:div">
      <s:assert test="count(tei:div) != 1">
        Si elle contient des subdivisions,
        une division doit en contenir
        au moins deux
      </s:assert>
    </s:rule>
  </constraint>
</constraintSpec>
```

Contraindre l'existence

Contraindre l'activation d'un élément ou d'un attribut en fonction d'un contexte donné

```
<constraintSpec ident="fromTo" scheme="schematron">
  <constraint>
    <s:rule context="tei:app[@type='structure']">
      <s:assert test="@from and @to">
        Le début et la fin d'un lemme doivent
        être identifiés
      </s:assert>
    </s:rule>
  </constraint>
</constraintSpec>
```


Contraindre le contenu

Contraindre le type de contenu d'une valeur d'attribut

```
<constraintSpec ident="fromVal" scheme="schematron">  
  <constraint>  
    <s:rule context="tei:app[@from]">  
      <s:assert test="matches(@from, '^#w\d+$')">  
        L'attribut @from doit contenir une  
        valeur qui commence par #w et finit  
        par un nombre  
      </s:assert>  
    </s:rule>  
  </constraint>  
</constraintSpec>
```

Tester les règles de validation

Pas d'erreur dans l'encodage \neq règle fonctionnelle


Il faut essayer de briser la règle pour voir si une erreur apparaît

Exercice

**Rédiger des règles
Schematron**



Consigne

- Reprendre l'ODD pour sonnetTEI.xml
- Rendre obligatoire la présence d'un ou plusieurs vers
- Un <lg type="sonnet"> doit commencer par un titre
- Paramétrer les <lg type="sonnet"> de telle sorte à ce qu'il contiennent deux quatrains et un sizain
- Écrire une règle schematron pour que les valeurs @n de <l> se suivent :
`number(@n) = number(preceding-sibling::tei:l[1]/@n) + 1`
 Attention, il y a un piège (<s:rule>)
- Générer le schéma RelaxNG et l'associer au fichier TEI