Séance 11

Les schémas XML

Personnalisation avancées de l’ODD

Construire une ODD

Procédé de création de la documentation et des spécifications d’un document TEI

Structure

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <-- MÉTADONNÉES -->
  </teiHeader>
  <text>
    <body>
      <div1>
        <-- DOCUMENTATION -->
      </div1>
      <div1>
        <schemaSpec ident="mon_ODD">
          <-- SPÉCIFICATIONS -->
        </schemaSpec>
      </div1>
    </body>
  </text>
</TEI>
```

Étapes

1) Encoder votre extrait
2) Produire une ODD avec ODD by example
3) Compléter et restructurer l'ODD avec une documentation et des spécifications
4) Produire un fichier Relax NG à partir de l'ODD modifiée
5) Produire une sortie HTML de son ODD

Consignes

Votre ODD doit contenir au moins :

- Une introduction de votre projet et ses exploitations possibles
- Une explication du fonctionnement de votre encodage et choix de balises
- Une règle contraignant l'usage d'un attribut et sa ou ses valeurs
- Une règle contraignant l'enchaînement de certains éléments
- Une règle contraignant la valeur d'un attribut ou l'usage d'un élément ou d'un attribut en fonction de son environnement

Bonnes pratiques

- Fichier indenté, propre, commenté au besoin
- Règles de validations ordonnées :
  - Modules en premier et classifiés
  - Règles modifiées plutôt vers le début
  - Règles inchangées de préférence vers la fin
- Ligne de commentaire expliquant la modification effectuée avant chaque règle modifiée

Ressources

Making a Unified ODD

Chapitre TEI sur la documentation

Chapitre TEI sur la spécification

Principaux éléments du module `tagdocs`

Documentation

`code`, `remarks`

Désignation des objets : `val`, `gi`, `att`, `tag`, `ident`, `altIdent`

Description de contenu : `valDesc`

Exemples : `eg`, `egXML`, `exemplum`

Description de spécifications : `specList`, `specDesc`

Documentation dans les spécifications

`gloss`, `desc`

Spécifications

Déclaration de spécification : `schemaSpec`, `elementSpec`, `classSpec`, `moduleSpec`, etc.

Référence : `classRef`, `macroRef`, `moduleRef`

Règles de contenu : `content`, `sequence`, `alternate`, `elementRef`, `textNode`, `anyElement`, `empty`

Règles d'attributs : `attDef` `attList`, `defaultVal`, `valItem`, `valList`, `dataSpec`, `datatype`, `dataRef`, `attRef`

Règles d'appartenance : `classes`, `memberOf`

Règles de validation (schematron) : `constraint`, `constraintSpec`

Documentation

Décrire ses choix d’encodage

Structure

```
<div1>
   <head>Titre de section</head>
   <p>Explication sur la section</p>
   <div2>
      <head>Titre de sous-section</head>
      <p>Explication sur la sous-section</p>
      <div3>
          <head>Titre de sous-sous-section</head>
          <p>Explication sur la sous-sous-section</p>
      </div3>
   </div2>
   <div2>[...]</div2>
</div1>
```

La documentation est placé dans une première `<div1>` est est structurée avec des éléments `div` allant jusqu'à `<div4>`. Chaque section est titrée avec un élément `<head>`.

Résultat de la transformation HTML

Table of contents

Désigner des éléments

```
<p>
    Les lettrines sont décrites dans le
    <gi>decoDesc</gi> et sont signalées dans
    le texte dans un élément <gi>g</gi> avec
    Pour valeur d'attribut <att>type</att>
    “<val>initiale</val>”.
</p>
```

La documentation est rédigée en prose courante mais les objets qu’elle mentionne sont balisés :

gi	Mention d’un élément
att	Mention d’un attribut
val	Mention d’une valeur d’attribut

Lister des éléments

```
<p>
    La mise en page originale est reproduite à l'aide
    des balises suivantes :
    <specList>
        <specDesc key="pb"/>
        <specDesc key="cb"/>
        <specDesc key="lb"/>
    </specList>
</p>
```

Donner des exemples

```
<p>Le neuf tironien est ainsi développé :</p>
<egXML xmlns="http://www.tei-c.org/ns/Examples">
    <choice>
        <abbr>&#42863;</abbr>
        <expan>con</expan>
    </choice>sellié
</egXML>
```

Les exemples sont placés dans des balises **<egXML>** avec l'espace de nom "http://www.tei-c.org/ns/Examples". Toute sorte d'exemples illustratifs peuvent être fournis dans le corps du texte avec **<eg>**.

Exercice

Ajouter un exemple à sa documentation

Consigne

- Générer une ODD pour `Le_Misanthrope_TEI.xml`
- Créer une documentation minimale avec au moins un exemple
- Générer la documentation HTML avec la transformation `TEI P5 XHTML`

Résultat de la transformation HTML

Ainsi, le neuf tironien est ainsi développé :

```
<choice>
 <abbr>9</abbr>
 <expan>con</expan>
</choice>sellié
```

Spécifications

Définir des règles de validation

Structure

```
<div1>
   <schemaSpec>
      <!-- Déclaration des modules utilisés ou non -->
      <moduleRef key="tei"/>
      <!-- Personnalisation des éléments -->
      <elementSpec>
          <constraintSpec>
              <!-- Règle de validation du contenu -->
          </constraintSpec>
          <attList>
              <!-- Règle de validation des attributs-->
          </attList>
      </elementSpec>
      <classSpec>
          <!-- Personnalisation des classes -->
      </classSpec>
      <classRef>
          <!-- Classe à inclure -->
      </classRef>
   </schemaSpec>
</div1>
```

Modifications possibles

Type

Suppression
Ajout
Modification
Remplacement

Objet

Éléments
Modèles de contenu
Attributs autorisés
Valeurs d’attribut
Classes

Quelques déclarations disponibles

`moduleSpec`

Documente la structure, le contenu et les fonctions d'un module

`moduleRef`

Référence un module qui doit être incorporé dans un schéma

`elementSpec`

Documente la structure, le contenu et l'usage d'un élément

`classSpec`

Documente le contenu d'une classe d'attribut ou d'un modèle de classe

Suppression d’un élément

La suppression d’élément est une *clean modification* sauf si c’est élément obligatoire en TEI (`<teiHeader>` ou tous les éléments de type `personLike` alors que `<listPerson>` est autorisé)

1) Suppression simple d'un élément avec `@mode="delete"`

```
<elementSpec ident="head" mode="delete"/>
```

2) Suppression d’élément ou classe dans un module avec `@except`

```
<moduleRef key="core" except="head"/>
```

3) Non insertion dans un module ou une classe avec `@include`

```
<moduleRef key="core" include="p author ~~head~~"/>
```

Ajout d’un élément

L’ajout d’élément n’est pas une pratique recommandée puisqu’elle ajoute au *namespace* TEI.

1) Ajout d'un élément avec `@mode="add"`

```
<elementSpec ident="alexandrin" mode="add"
             ns="http://www.example.org/ns/nonTEI">
    <classes>
       <memberOf key="model.lLike"/>
       <memberOf key="macro.paraContent"/>
    </classes>
    <content><textNode/></content>
</elementSpec>
```

2) Déclaration d'un élément dans le `<moduleRef>` avec `@include`

```
<moduleRef key="textstructure" include="alexandrin"/>
```

Modification d’un élément

La modification d’un élément peut porter sur son contenu :

- contrainte d’enchaînement d’éléments précis
- typage des données permises
- attributs autorisés
- valeurs d’attribut valides

Modification d'un élément avec `@mode="change"`

```
<elementSpec ident="lg" mode="change">
    [...]
</elementSpec>
```

Suppression d’attribut

```
<elementSpec ident="seg" mode="change">
   <gloss>Segment de texte</gloss>
   <attList>
      <attDef ident="corresp" mode="delete"/>
   </attList>
</elementSpec>
```

Ajout d’attribut

```
<elementSpec ident="seg" mode="change">
   <gloss>Segment de texte</gloss>
   <attList>
      <attDef ident="length" mode="add"
              ns="http://www.exemple.com/ns/nonTEI"/>
   </attList>
</elementSpec>
```

Restriction des valeurs d’attribut

```
<elementSpec ident="lg" mode="change">
    <attList>
        <attDef ident="type" mode="change">
            <valList mode="add" type="closed">
                <valItem ident="quatrain"/>
                <valItem ident="sizain"/>
                <valItem ident="sonnet"/>
                <valItem ident="tercet"/>
            </valList>
        </attDef>
    </attList>
</elementSpec>
```

Exercice

Ajouter un attribut et définir ses valeurs

Consigne

- Créer une ODD pour `sonnetTEI.xml`
- Déclarer dans l'ODD du texte de Verlaine un attribut "`rime`" (non TEI) pour l'élément `<l>`
- Créer une liste de valeurs close :
  "`rime A | rime B | rime C`"
- Transformer en `.rng`
- Associer à l'encodage TEI

Typage de valeurs d’attribut

Typage avec `datatype` et son enfant `dataRef` avec l'attribut `@key` si on pointe vers un [type de données défini par la TEI](#) (`@name` pour XML schéma ou RelaxNG) :

```
<attList>
   <attDef ident="type" mode="change">
      <desc xml:lang="fr">
         L'attribut @next doit contenir une
         référence à un xml:id
      </desc>
      <datatype>
         <dataRef key="teidata.pointer"/>
      </datatype>
   </attDef>
</attList>
```

Occurrence des valeurs d’attribut

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

```
<attList>
   <attDef ident="type" mode="change">
      <desc xml:lang="fr">
         L'attribut @next doit contenir une
         ou plusieurs références à un xml:id
      </desc>
      <datatype minOccurs="1" maxOccurs="unbounded">
         <dataRef key="teidata.pointer"/>
      </datatype>
   </attDef>
</attList>
```

Statut des valeurs d’attribut

La valeur par défaut de `@usage` est “`opt`” (optionnelle) et elle peut prendre pour valeur “`rec`” (recommandée) ou “`req`” (requise) :

```
<attList>
   <attDef ident="next" mode="change" usage=”req”>
      <desc xml:lang="fr">
         L'attribut @next est obligatoire et doit
         contenir une référence à un xml:id
      </desc>
      <datatype>
         <dataRef key="teidata.pointer"/>
      </datatype>
   </attDef>
</attList>
```

Exercice

Restreindre les valeurs des attributs

Consigne

- Reprendre l'ODD pour `sonnetTEI.xml`
- Typer la valeur d'un attribut
- Limiter les occurrences d'un attribut
- Déclarer un attribut obligatoire
- Transformer en `.rng`
- Associer à l'encodage TEI

Documenter un élément/attribut

```
<elementSpec ident="msIdentifier" mode="change">
    <gloss>Identifiants du manuscrit</gloss>
    <desc>
        contient les différents identifiants associés au
        manuscrit : cotes actuelles et anciennes
        (idno et altIdentifier), lieu (repository) et
        pays de conservation (country).
    </desc>
</elementSpec>
```

Il est possible de donner des informations supplémentaires sur un élément ou attribut dans les spécifications :

`gloss` Désignation comme on pourrait la trouver dans un glossaire

`desc` Définition de l'usage de l'objet et de son contenu

Exercice

Documenter une spécification d’élément

Consigne

- Reprendre l'ODD `Le_Misanthrope_TEI.xml`
- Ajouter `<gloss>` et `<desc>` à un `<elementSpec>`
- Ajouter `<gloss>` et `<desc>` à un `<attDef>`
- Générer la documentation HTML avec la transformation `TEI P5 XHTML`

Résultat de la transformation HTML

<msDesc>

Description du manuscrit contient les détails concernant le manuscrit à l'origine de l'encodage : ses identifiants (msIdentifier), son titre (head), son contenu (msContents), sa description matérielle (physDesc) et son histoire (history).	
@~~corresp~~	
@~~next~~	
@~~ana~~	
@~~facs~~	
@~~resp~~	
@~~source~~	
@~~type~~	

Définir une séquence de contenu

`<content>` apparaît toujours avant `<attList>` dans l'`<elementSpec>` : il peut contenir un élément `<sequence>` qui définit un enchaînement d'élément. Son attribut `@preserveOrder` permet de spécifier si l'ordre de déclaration est signifiant

```
<elementSpec ident="div" mode="change">
   <content>
      <sequence preserveOrder="true">
        <elementRef key="head" minOccurs="1"
                    maxOccurs="1"/>
        <elementRef key="p" minOccurs="1"
                    maxOccurs="unbounded"/>
     </sequence>
  </content>
</elementSpec>
```

Définir un contenu textuel

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

<elementSpec ident="p" mode="change">
   <content>
      <textNode/>
  </content>
</elementSpec>
```

Définir une alternance de contenu

```
<elementSpec ident="choice" mode="change">
   <content>
      <alternate>
         <sequence>
            <elementRef key="sic"/>
            <elementRef key="corr"/>
         </sequence>
         <sequence>
            <elementRef key="orig"/>
            <elementRef key="reg"/>
         </sequence>
      </alternate>
   </content>
</elementSpec>
```

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">
    [...]

    <s:assert test="tei:div">[...]</s:assert>

</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>
```

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>
```

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 quatrain 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
- Générer le schéma RelaxNG et l'associer au fichier TEI