Comment maîtriser le tigre TEI

Lou Burnard Consulting

Comment maitriser le tigre TEI ?

À quoi ça sert, un ODD ?

- ODD est le langage de définition et de maintenance du système TEI
- Il permet le maintenance du code et de sa documentation d'une manière intégrée, à partir d'une seule source XML
- Il s'applique également à la définition d'un système de balisage non-TEI, mais il est d'habitude utilisé pour la définition des personnalisations ou des modifications de la TEI.
- Dans les deux cas, ODD fournit une manière efficace d'assurer la perennité de vos données, en vous obligeant de documenter leur usage d'une manière standardisée

ODD sert aussi à explorer les richesses de la TEI ...

Les deux côtés d'un ODD

Nous avons besoin de deux choses complémentaires…

Un schéma formel (utilisant un langage informatique tel que DTD, RELAX NG, W3C Schema, Schematron) pour contrôler l'édition

- déterminer quelles sont les balises disponibles ?
- dans quels contextes ?
- avec quels attributs ?
- avec quelles valeurs ?
- en respectant quelles contraintes ?

Une documentation pour expliciter aux utilisateurs/développeurs nos principes éditoriaux, nos principes de choix de balises, etc. :

- dans plusieurs langues naturelles ;
- dans plusieurs formats de fichier (PDF, MsWord, HTML, epub,...).

OK, mais pourquoi ODD ?

Ces attentes pourraient être satisfaites de plusieurs manières.

Les avantages d'ODD :

- un format XML bien établi,
- faisant partie intégrante du système TEI,
- permettant un traitement fortement intégré avec d'autres systèmes TEI,
- pérennisation à long terme
- standardisation

L'idée essentielle de ODD (1)

One Document Does it all

Un vocabulaire spécialisé pour la déclaration et la documentation :

- des types d'élément XML, indépendants des schémas
- des patrons (macros)
- des types de donnée (datatype)
- des classes (et sous-classes) d'éléments
- des regroupements de telles déclarations (specGrp)
- et des schémas

Un schéma peut intégrer :

- des objets choisies dans la liste ci-dessus
- des références à un de ces objets
- des références a un regroupement prédéfini de tels objets (un <moduleRef>)

L'idée essentielle de ODD (2)

Un ODD peut combiner plusieurs spécifications pour un même objet

- une qui est 'canonique', référencée dans les *Guidelines*
- une (ou plusieurs) supplémentaires, modifiées en partie ou totalement
- ces versions partielles peuvent être explicitées dans l'ODD ou bien héritées d'un autre ODD
- le statut de chaque spécification supplémentaire est explicité par son attribut *@mode*, par defaut `add`

ODD est surtout un langage de specification...

La TEI s'exprime en langage ODD. Ce fut d'ailleurs la raison principale pour laquelle le langage fut inventé

La source TEI P5 (disponible ici `http://www.tei-c.org/release/xml/tei/odd/Source/`) rassemble :

- 39 fichiers en TEI-XML, dont 25 contiennent un chapitre de documentation en XML-TEI, la plupart définissant un module, par exemple PH-PrimarySources.xml
- 832 fichiers en TEI-XML, chacun définissant un élément, une classe, un type de donnée, ou une macro
  - 33 types de donnees (teidata.xxxx) par ex. teidata.percentage
  - 118 classes de type modèle (code>model.xxxx) par ex. model.biblLike.xml
  - 73 classes de type attribut (`att.xxxx`) par ex. att.divLike.xml
  - 36 macros (macro.xxxx) par ex. macro.phraseSeq.xml
  - 567 spécifications d'éléments, de ab.xml jusqu'à zone.xml

... mais ODD est aussi un langage de personnalisation

On se sert du même système pour spécifier ses choix dans le grand bazar de la TEI et pour spécifier le bazar lui-même.

Un ODD de personnalisation est spécifié par rapport à un autre ODD ; typiquement mais pas forcément celui qui définit la version actuelle des *Guidelines*

- en sélectionnant des modules
- en sélectionnant parmi les objets (éléments, classes, datatypes, macros) fournis par ces modules
- en supprimant ou modifiant quelques uns des attributs fournis par ces objets
- en modifiant ou remplaçant quelques parties de ces spécifications (par ex. les valList)
- éventuellement en ajoutant des spécifications d'objets nouveaux

Comment créer un ODD

Un ODD, vous le savez déjà, c'est un document TEI comme les autres. Vous pouvez le construire et modifier avec oXygen ou n'importe quel autre éditeur XML

Il existe des outils pour faciliter une construction initiale :

- http://www.tei-c.org/Roma; voir aussi tutoriel introductif (en anglais)
- ODD By Example

L'élément <schemaSpec>

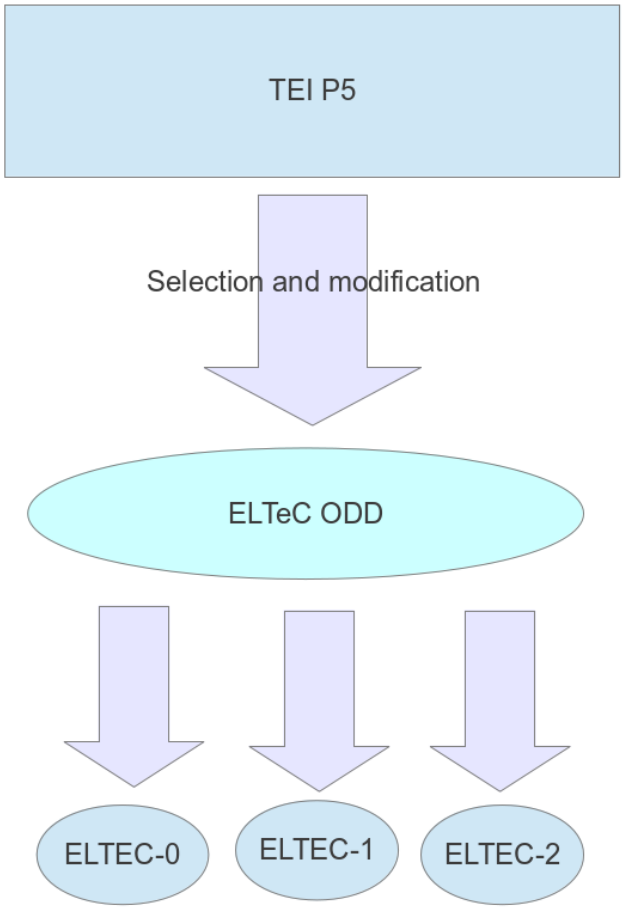
On utilise l'élément <schemaSpec> pour spécifier un schéma

- L'attribut @*ident* obligatoire fournit un nom pour le schéma
- L'attribut @*start* indique le ou les noms des élément(s) racine(s) du schéma
- L'attribut @*source* indique l'emplacement des déclarations référencées par le schéma (par ex une version spécifique de TEI P5)
- Les attributs @*docLang* et @*targetLang* permettent la sélection des langues à utiliser pour les descriptions d'éléments et pour les noms d'élément respectivement, en supposant la présence dans cette source des traductions requises

```
<schemaSpec start="TEI"
  ident="testschema" source="tei:1.5.0" docLang="fr">
<!-- declarations -->
</schemaSpec>
```

Astuces

- Un seul document peut rassembler plusieurs <schemaSpec> qui partagent (par ex) un ensemble commun de <specGrp>
- Un ODD peut servir comme base pour un autre (‘oddChaining’)



Usage de l'attribut *@source*

L'attribut *@source* sert à spécifier la source des déclarations qu'on souhaite intégrer : par défaut dans la version la plus récente de TEI P5.

Ou bien…

```
<schemaSpec ident="test2"
 prefix="tei_" start="TEI" source="tei:1.5.0">
<!-- dans la version 1.5.0 -->
</schemaSpec>
```

```
<schemaSpec ident="test4"
 source="myCompiled.odd">
<!-- ensemble de déclarations maison -->
</schemaSpec>
```

@source peut aussi être utilisé sur <classRef>, <elementRef>, <macroRef>, et <moduleRef> : il doit pointer vers un ODD compilé

Composants d'un <schemaSpec>

- <elementSpec>, <classSpec>, <dataSpec>, <macroSpec> (etc.) : des objets nouveaux
- <elementRef>, <classRef>, <dataRef>, <macroRef>(etc.) : des objets déjà existants quelque part
- <moduleRef> : un ensemble d'objets fournis par un module TEI

L'element moduleRef permet la sélection d'un ensemble d'objets TEI; par défaut dans son entièreté

La location des objets référenciés est specifiée par un attribut *@source* : par défaut il s'agit de la version la plus récente de TEI P5

moduleRef: sélection par exclusion

Vous pouvez spécifier les éléments que vous souhaitez supprimer parmi ceux proposés par un module :

```
<schemaSpec start="TEI"
 ident="testschema">
 <moduleRef key="core"
  except="mentioned said"/>
<!-- ... -->
</schemaSpec>
```

ou également :

```
<schemaSpec start="TEI"
 ident="testschema">
 <moduleRef key="core"/>
 <elementSpec ident="mentioned"
  mode="delete"/>
 <elementSpec ident="said"
  mode="delete"/>
<!-- ... -->
</schemaSpec>
```

(L'attribut @*mode* contrôle la résolution de déclarations multiples

moduleRef : sélection par inclusion

Vous pouvez spécifier les éléments que vous souhaitez inclure parmi ceux qui sont proposés par un module :

```
<schemaSpec start="TEI"
 ident="testschema">
 <moduleRef key="textstructure"
  include="body div"/>
<!-- ... -->
</schemaSpec>
```

ou également :

```
<schemaSpec start="TEI"
 ident="testschema" source="tei:5.1.2">
 <elementRef key="div"/>
 <elementRef key="body"/>
<!-- ... -->
</schemaSpec>
```

Attention ! un module peut définir d'autres choses que des éléments. Les attributs *@include* et *@except* ne s'appliquent qu'au éléments

Usage des module

- Vous n'êtes pas obligé de préciser son module si vous souhaitez préciser l'inclusion d'un élément !
- Un <elementRef> suffit – pourvu que l'élément concerné soit déclaré par la source invoquée
- Mais il faut préciser son module si vous souhaitez inclure/utiliser une classe de modèle ou une classe d'attribut.
- La plupart des classes TEI étant définies par le module tei, ce dernier est à peu près essentiel pour tout ODD TEI.

Spécifications multiples

Le traitement d'un ODD implique la résolution de spécifications multiples pour un même objet

Supposons deux <elementSpec> pour un même élément, la résolution est déterminée par la valeur de l'attribut *@mode*

- `mode='add'` (défaut) : effectuer une nouvelle déclaration
- `mode='delete'` : aucune déclaration n'est valide, l'élément est supprimé
- `mode='replace'` : cette déclaration remplace entièrement toute autre déclaration
- `mode='change'` : les parties de cette déclaration remplacent les parties correspondantes dans une autre déclaration ; le reste ne change pas.

Par exemple

```
<schemaSpec start="TEI"
 ident="testschema">
 <moduleRef key="core" include="p hi"/>
 <elementSpec key="p" mode="delete"/>
</schemaSpec>
```

L'élément <p> est supprimé

```
<schemaSpec start="TEI"
 ident="testschema">
 <moduleRef key="core" include="p hi"/>
 <elementSpec key="p" mode="change">
  <desc>Cet élément ne contient que de texte</desc>
  <content>
   <textNode/>
  </content>
 </elementSpec>
</schemaSpec>
```

Le contenu et la description de l'élément <p> sont changés ; ses autres propriétés (par ex, ses attributs) ne changent pas

Traitement d'un ODD

- Toutes les déclarations (*Spec et *Ref) sont d'abord réunies et canonicalisées
- Les déclarations multiples sont résolues
- La sortie de cette procédure est ensuite transformée en schéma, et/ou en documentation

La feuille de style odd2odd nous permet d'effectuer les deux étapes initiales et d'en conserver le résultat : nous appelons cela un *compiled ODD*

Un tel fichier est réutilisable comme la *@source* d'un autre ODD

Regardons un exemple

```
<body>
  <head>Une personalisation TEI pour la
transcription collaborative</head>
  <p>Cette personalisation propose un
schéma minimal pour la transcription
collaborative
     des documents archivals. </p>
  <schemaSpec ident="transMin"
    start="TEI text div" docLang="fr">
    <moduleRef key="tei"/>
    <moduleRef key="header"
      include="teiHeader fileDesc
titleStmt publicationStmt sourceDesc"/>
    <moduleRef key="textstructure"
      include="TEI text body div"/>
    <elementRef key="ab"/>
    <elementRef key="pb"/>
    <elementRef key="unclear"/>
    <elementRef key="hi"/>
    <elementRef key="name"/>
    <elementRef key="title"/>
    <classRef key="att.global.rendition"
      except="rendition style"/>
    <classSpec type="atts"
      ident="att.declaring" mode="delete"/>
    <classSpec type="atts"
      ident="att.edition" mode="delete"/>
    <classSpec type="atts"
      ident="att.editLike" mode="delete"/>
  </schemaSpec>
</body>
```

Cette personalisation rassemble :

- un peu de documentation
- un <schemaSpec> identifiable, précisant une langue de documentation et des éléments racine
- une séléction d'éléments a inclure
- la suppression de plusieurs attributs

Exemple cont

```
<body>
  <head>Une personalisation TEI pour la
transcription collaborative</head>
  <p>Cette personalisation propose un
schéma minimal pour la transcription
collaborative
     des documents archivals. </p>
  <schemaSpec ident="transMin"
    start="TEI text div" docLang="fr">
<!-- ... -->
    <elementSpec ident="botName"
      ns="http://monexcellentprojet.com">
      <desc>nom botanique</desc>
      <desc xml:lang="en">botanical
name</desc>
      <classes>
        <memberOf key="model.phrase"/>
        <memberOf key="att.global"/>
      </classes>
      <content>
        <macroRef key="macro.paraContent"/>
      </content>
    </elementSpec>
<!-- ... -->
  </schemaSpec>
</body>
```

- nous ajoutons une spécification pour un élément non-TEI, appartenant à une autre espace de nommage
- cette spécification comporte
  - une description
  - une indication des classes TEI auxquelles l'élément appartiendrait
  - une indication de son contenu possible

Exemple cont

```
<body>
  <head>Une personalisation TEI pour la
transcription collaborative</head>
  <p>Cette personalisation propose un
schéma minimal pour la transcription
collaborative
     des documents archivals. </p>
  <schemaSpec ident="transMin"
    start="TEI text div" docLang="fr">
<!-- ... -->
    <elementSpec ident="hi"
      mode="change">
      <attList>
        <attDef ident="rend"
          mode="replace">
          <datatype>
            <dataRef key="teidata.enumerated"/>
          </datatype>
          <valList type="closed">
            <valItem ident="underline"/>
            <valItem ident="superscript"/>
          </valList>
        </attDef>
      </attList>
    </elementSpec>
<!-- ... -->
  </schemaSpec>
</body>
```

- la spécification existante pour <hi> est modifiée
- la spécification de son attribut *@rend* est remplacée:
  - son type de données devient une énumération
  - les valeurs possibles sont explicités

Structuration d'un ODD

Mis à part le <schemaSpec> qui définit le schéma, on peut organiser le document comme tout autre document TEI, utilisant , <list>, etc.

Au sein de ce document, des éléments supplémentaires sont prévus pour le regroupement des déclarations à l'extérieur du <schemaSpec> :

- <specGrp> : un regroupement de déclarations identifiable

```
<p>Nous n'utilisons que ces éléments du module
<ident>linking</ident> : <specGrp xml:id="linkingElts">
<!-- quelques elementSpec etc -->
 </specGrp>
</p>
```

- <specGrpRef> : indique où on souhaite intégrer les déclarations précisées par un <specGrp>

```
<specGrpRef target="#linkingElts"/>
```

Trois sujets avances

- <constraintSpec> et <dataSpec>: contraintes sur le contenu plus complexes
- <model> : specification de traitement par defaut
- Addition des composants d'une schéma non-TEI

L'élément <dataSpec>

Definit un *type de données* applicable habituellement à un ou plusieurs attribut.

- la TEI ne réinvente pas les types de données (datatypes) déjà fournis par le W3C
- mais elle permet l'ajout de quelques modifications, notamment
  - des *facets* (par ex 'valeur integral entre 4 et 42')
  - des *patterns* (par ex 'chaine de caracters conforme à ce regexp'
  - et une surcouche sémantique (par ex, `teidata.sex` vs `teidata.enumerated`)

Exemples dataspec

```
<dataSpec ident="teidata.pointer">
  <content>
    <dataRef name="anyURI"/>
  </content>
</dataSpec>
```

```
<dataSpec ident="myAngle">
  <content>
    <dataRef name="decimal">
      <dataFacet name="maxInclusive"
        value="360.0"/>
      <dataFacet name="minInclusive"
        value="-360.0"/>
    </dataRef>
  </content>
</dataSpec>
```

```
<dataSpec ident="data.percentage">
  <content>
    <dataRef name="nonNegativeInteger"
      restriction="[0-9][0-9]?%"/>
  </content>
</dataSpec>
```

Contraintes de données avec Schematron

- Une spécification d'élément peut proposer des contraintes supplémentaires sur son contenu en utilisant un ou plusieurs éléments <constraintSpec>
- Ces règles sont exprimées en utilisant le langage ISO Schematron

```
<elementSpec ident="div"
 module="teistructure" mode="change"
   xmlns:s="http://purl.oclc.org/dsdl/schematron">
 <constraintSpec ident="div">
  <constraint>
   <s:assert test=".//tei:p">une division doit contenir au
moins un
        paragraphe</s:assert>
  </constraint>
 </constraintSpec>
</elementSpec>
```

L'élément <constraintSpec>

Il définit une contrainte qui s'applique au sein de l'élément dans lequel il est déclaré

- L'attribut *@ident* est obligatoire : il fournit un identifiant unique
- Il rassemble un ou plusieurs <constraint>
- L'élément <constraint> contient (typiquement) un <assert> ou un <report>, éléments de l'espace de nommage `http://purl.oclc.org/dsdl/schematron`

Fonctionnement des règles Schematron

- Le contenu de l'élément <assert> est affiché si le test est false
- Le contenu de l'élément <report> est affiché si le test est true
- Astuce : plusieurs éléments schematron sont disponibles pour enrichir le texte du message affiché, notamment <name> (context) et <value-of> (valeur)

Un schéma RNG intégrant ces règles sera auto-généré si l'on utilise le logiciel oXygen pour traiter son ODD

Applications typiques des règles Schematron

- Contraintes de co-occurrence : ‘si l’attribut X a la valeur A, l’élément qui le porte doit contenir un Y’
- Contraintes arithmétique contextuelles : ‘au sein d’un <titleStmt>, on ne permet qu’un seul <title>’
- Contraintes textuelles : ‘Les caractères ’ et “ ne sont pas permis au sein d’un <p> apparaissant dans le <body>’
- Contraintes contextuelles : ‘mots en français (@*xml:lang*=’fr’) ne sont pas permis au sein d’un élément latin (@*xml:lang*=’la’)’
- Intégrité référentielle : ‘un pointer exprimé sous la forme d’une URL et commençant par # doit correspondre à un élément ayant un @*xml:id* identique quelque part dans le document’

Par exemple…

```
<constraintSpec ident="isoconstraint">
 <constraint>
  <s:assert test="tei:fileDesc/tei:titleStmt/tei:title[@type='mai
il faut fournir
      un titre principal </s:assert>
 </constraint>
</constraintSpec>
```

```
<elementSpec ident="figure">
<!-- ... -->
 <constraintSpec ident="demo-c2">
  <constraint>
   <s:report test="not(tei:figDesc or tei:head)"> Votre
figure ne contient ni un
        figDesc ni un head : aucun attribut <att>alt</att>
n'est générable</s:report>
  </constraint>
 </constraintSpec>
</elementSpec>
```

Ajout d'un 'processing model' (modèle de traitement)

Vous pouvez également enrichir votre documentation avec des déclarations plus précises sur la manière dont des éléments particuliers devraient être mis en forme.

Cela est complémentaire aux fonctionnalités offertes avec *@rend* et *@style* qui décrivent la manière dont la source originale (non-digitale) a été formattée.

```
<elementSpec mode="change"
 ident="quote">
 <model predicate="ancestor::p"
  behaviour="inline"/>
 <model predicate="not(ancestor::p)"
  behaviour="block"/>
</elementSpec>
```

Process a <quote> inside a <p> as an inline; elsewhere as a block

Un <model> precise le/s 'behaviour/s' attendu/s

- Un behaviour peut correspondre à un des concepts de formattage employés communément, par exemple, 'block', 'inline', 'pointer'
- ou à une traîtement de niveau plus haut, par exemple, 'alternate'

behavic	params	function
alternate	(default, alternate)	Supporte la présentation de visualisations alternatives, par exemple en rendant le contenu préféré, en le présentant en parallèle, ou en bien en permettant de passer de l'un à l'autre.
graphic	(url, width, height, scale, title)	si url est présent, l'utiliser pour rendre l'élément graphic, sinon rendre une image placeholder.
omit		ne rien faire, ne pas traiter les enfants
inline	(content, label	créer un élément inline

Voir implementation plus ou moins complète : TEI Publisher

Addition des composants d'une schéma nonTEI

On souhaite utiliser l'élément TEI <formula> et y insérer du contenu exprimé en MathML

Il nous faut donc :

1. inclure les composants du schéma MathML
2. modifier le modèle de contenu de l'élément <formula>
3. générer un schéma qui résout les conflits de nommage

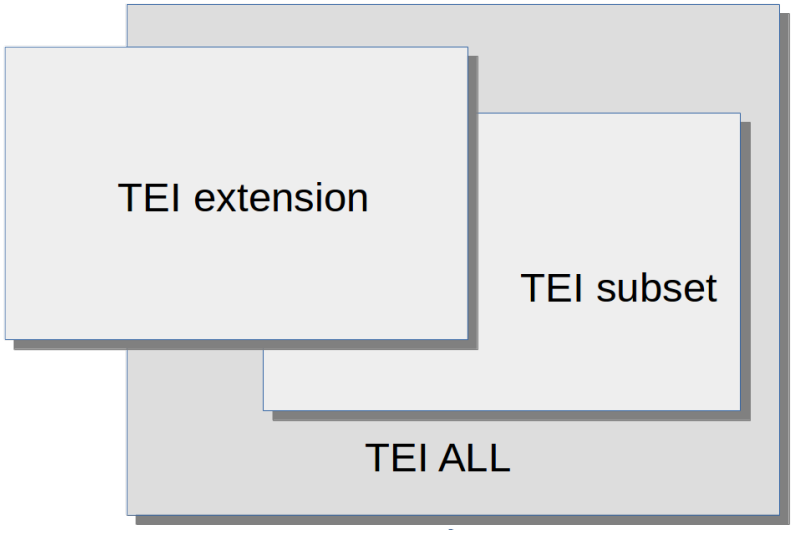
ATTENTION : il y a un élément <list> dans TEI mais également dans MathML !

TEI + MathML : le ODD

```
<schemaSpec ident="tei_math"
 prefix="tei_" start="TEI teiCorpus">
 <moduleRef url="http://www.tei-c.org/release/xml/tei/
custom/schema/relaxng/mathml2-main.rng"/>
 <moduleRef key="header"/>
 <moduleRef key="core"/>
 <moduleRef key="tei"/>
 <moduleRef key="textstructure"/>
 <moduleRef key="figures"/>
 <elementSpec module="figures"
  ident="formula" mode="change">
  <content>
   <elementRef key="mathml.math"/>
  </content>
 </elementSpec>
</schemaSpec>
```

L'attribut *@prefix* nous permet de désambiguïser les identifiants ressortant des schémas différentes

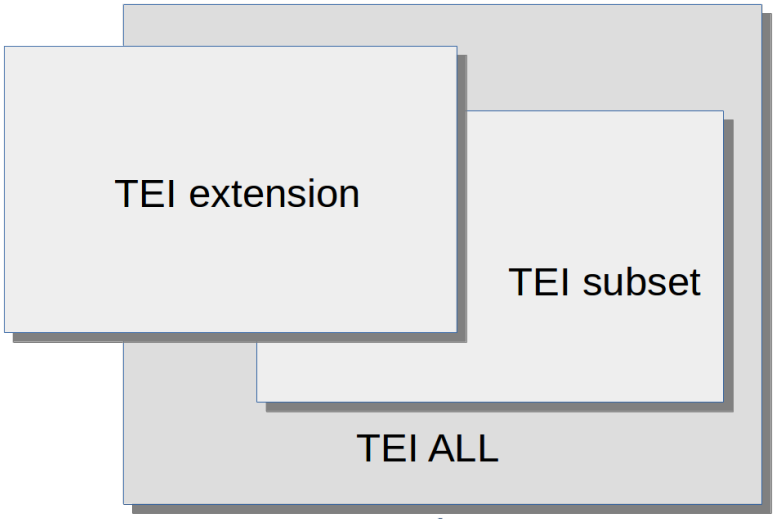
Variétés de ODD



Chacune de ces formes représente :

- un ODD
- le schéma généré à partir de cet ODD
- l'ensemble de documents considérés comme valide par ce schéma

Variétés de ODD



- un ‘TEI subset’ fournit un sous-ensemble des composants constituant TEI All
- un ‘TEI extension’ contient des composants qui ne font pas partie de TEI All

La question $1000k : comment savoir si votre extension serait conforme à la TEI ?

Le plus simple

- Pour un TEI subset : vos modifications génèrent un schéma réduit plus précis, plus adapté à votre projet et une documentation plus exacte et correcte pour votre communauté d'utilisateurs
- Vos documents restent toujours valides par rapport à TEI All et vous respectez toujours le modèle sémantique de la TEI.
- Pour un TEI extension : les extensions éventuelles sont signalées clairement, en utilisant une autre espace de nommage, et sont d'ailleurs expliquées dans votre ODD, par exemple en employant les classes TEI, l'element <equiv>, la documentation, etc.

Qu'est-ce signifie « être conforme à la TEI » ?

- être honnête : Les éléments XML qui sont déclarés comme appartenant au namespace TEI doivent respecter les définitions TEI de ces éléments
- être explicite : Pour valider un document TEI, un ODD est fortement conseillé, parce que cela mettra en évidence toutes les modifications effectuées.

Validation d'un document par rapport à un schéma TEI (TEI All, subset, ou extension) est un bon signe, mais ne garantit pas son conformance

Le 'respect des definitions TEI' implique un control semantique pas automisable

Les limites de la modification

- Est-ce que l'on peut supprimer n'importe quoi ? par ex. <title> ?
- A quoi servent les classes vides ?
- Est-ce qu'on peut ajouter n'importe quoi ? par ex. elements du Dublin Core

- L'objet des règles de conformité est de simplifier le 'blind interchange' des documents ; mais elles ne le garantissent pas.
- L'enjeu est de permettre à un utilisateur de comprendre votre encodage, non pas forcement de le contraindre à vous suivre aveuglement

Pour aller plus loin

- tutoriel pratique sur l'enchainement des ODD
- tutoriel pratique sur la generation des ODD
- tutoriel pure ODD (en anglais)
- le manuel: TEI Guidelines, chap 22
- tutoriel introductif sur l'usage de Roma pour creer un ODD (en anglais)
- n'hesitez pas de posez vos questions sur TEI-L ou tei-fr