

One Document Does it all: le système ODD

TEI@Oxford

novembre 2012

L'application d'un balisage 'intelligente'

La TEI s'applique à l'encodage des ...

- composants structuraux et fonctionaux d'un texte
- transcriptions diplomatique des sources historiques, des images, des annotations
- lien, correspondences, alignements
- données et entités : par ex temps, personnes, lieux, evenements
- annotations peritextuelles et metatextuelles (correction, suppression, ajouts)
- analyses linguistiques
- metadonnées de plusieurs types
- ... et des définitions formelles de schéma XML !

Cette intervention concerne le dernier.

Pourquoi ODD?

- On a besoin d'un schéma XML pour définir ses ressources
- On desire fournir de
 - documentation de la sémantique de son schéma XML
 - notices des contraintes appliqués, des notes d'usage, des exemples
- Vous devez les maintenir en l'étape!
- Vous souhaitez partager cette documentation
 - avec d'autres personnes
 - avec vous-meme, à long terme
- Et vous préférez ne pas réinventer la roue...

ODD est né en 1991, a Bergen, Norvege



Lido Cafe, Torgallmenningen 1a, Bergen

"A charming place with great location"



Reviewed 19 March 2012 via mobile

Nice food and lovely people there. The table cloth is replaced with two big sheets of blank paper and you can doodle whatever you like. The waitress was very patient and friendly. We all loved it!

Was this review helpful? ☒ Yes

[Problem with this review?](#)

ODD n'était pas un nouvel concept



ODD : ca aurait pu être RDF...

- "iterate programming" (Donald Knuth)
- java beans, doxygen...

L'idée essentielle (1)

Un vocabulaire spécialisé pour la définition

- des schémas
- des types d'élément XML, indépendant de schéma
- des regroupements de tels éléments, publiques ou privées
- des patrons (MLE macros)
- des classes (et sousclasses) d'éléments

Pour la définition également des références, qui permettront de réunir dans un schéma

- des objets identifiables (dans la liste ci-dessus)
- des objets appartenant à d'autres espaces de nom

et qui serait intégrable avec un système de balisage documentaire classique

L'idée essentielle (2)

Un processeur ODD :

- rassemble les composants référencés ou fournis
- résoud les incohérences ou duplications eventuelles
- peut controller ou valider les résultats
- émet une schéma exprimée en langue formelle (RELAXNG, DTD, WSD...)
- émet un document XML "pure" avec les composants documentaires

<http://www.tei-c.org/Roma>

Premier exemple simplissime

Nous avons besoin de `<stuff>`, qui contient un mélange des `<bit>`s et de `<bob>`s. Nous ne connaissons rien au sujet de la TEI, et n'en avons pas envie. Pareil pour les espaces de noms.

```
<schemaSpec ns="" start="stuff" ident="simpleS">
  <elementSpec ident="stuff">
    <desc>élément racine d'un schéma simplissime</desc>
    <content>
      <rng:oneOrMore>
        <rng:choice>
          <rng:ref name="bit"/>
          <rng:ref name="bob"/>
        </rng:choice>
      </rng:oneOrMore>
    </content>
  </elementSpec>
  <!-- ... continue au prochain transparent -->
</schemaSpec>
```


exemple simplissime, contd.

```
<schemaSpec ns="" start="stuff" ident="simpleS">
<!-- ... contd -->
  <elementSpec ident="bob">
    <desc>élément pointeur du schéma simplissime</desc>
    <content>
      <rng:empty/>
    </content>
    <attList>
      <attDef ident="href">
        <desc>fournit l' URI de l'objet cible</desc>
        <datatype>
          <rng:data type="anyURI"/>
        </datatype>
      </attDef>
    </attList>
  </elementSpec>
  <elementSpec ident="bit">
    <desc>élément textuel du schéma simplissime (peut contenir des bobs) </desc>
    <content>
      <rng:zeroOrMore>
        <rng:choice>
          <rng:text/>
          <rng:ref name="bob"/>
        </rng:choice>
      </rng:zeroOrMore>
    </content>
  </elementSpec>
</schemaSpec>
```

So what?

- On peut maintenant générer un schéma RELAXNG, W3C schéma, or DTD language par une transformation XSLT
- On peut extraire les fragments documentaires, notamment les descriptions des éléments and des attributs

TEI fournit une élément spécialisé pour cela:

```
<specList>
  <specDesc key="bit"/>
  <specDesc key="bob" atts="href"/>
</specList>
```

qui dans le document XML de sortie serait transformé en

```
<list type="gloss">
  <label>
    <gi>bit</gi>
  </label>
  <item>élément textuel de schéma simplissime (peut contenir des bobs) </item>
  <label>
    <gi>bob</gi>
  </label>
  <item>élément pointeur du schéma simplissime</item>
</list>
```

Qu'est-ce qu'on pourrait souhaiter dire à propos des éléments?

- `<desc>`s ou `<gloss>` en plusieurs langues
- exemples d'usage
- constraints additionels (schématron)
- documentation des listes des valeurs closes
- association avec TEI classes

Exemple de descriptions alternatives

```
<elementSpec module="core" ident="p">
<gloss>paragraph</gloss>
<gloss version="2007-12-20" xml:lang="kr">문단</gloss>
<gloss version="2007-05-02" xml:lang="zh-tw">□□</gloss>
<desc>marks paragraphs in prose.</desc>
<desc version="2007-12-20" xml:lang="kr">산문에서 문단을 표시한다.</desc>
<desc version="2007-05-02" xml:lang="zh-tw">□□□□□□□□</desc>
<desc version="2008-04-05" xml:lang="ja"> 散文の段落を示す. </desc>
<desc version="2009-01-06" xml:lang="fr">marque les paragraphes dans un
texte en
prose.</desc>
<desc version="2007-05-04" xml:lang="es">marca párrafos en prosa.</desc>
<desc version="2007-01-21" xml:lang="it">indica i paragrafi in
prosa</desc>
<!-- ... -->
</elementSpec>
```

Exemples d'usage

L'élément `<exemplum>` contient un exemple de son usage et une note la-dessus...

```
<exemplum xml:lang="en">
  <egXML><egXML><langUsage>
    <language ident="en">English</language>
  </langUsage>
</egXML>
</egXML>
<p>In the source of the TEI Guidelines, this élément declares itself and its content
  as belonging to the namespace <ident type="ns">http://www.tei-c.org/ns/Examples</ident>.
  This enables the content of the élément
  to be validated independently against the TEI scheme.</p>
</exemplum>
```

Définition du contenu d'un élément

- Actuellement, la TEI se sert du langage RELAXNG pour définir le contenu ("content model") des éléments et des attributs
- (mais on à propose de modifier cela)

Les constraints sont donc exprimables de plusieurs manières :

- par reference à un `<valList>` élément ...
- par reference à un élément `<datatype>` élément (s'applique uniquement aux attributs)
- par inclusion des `<constraintSpec>` éléments (expression en ISO schématron)

OK, on revient sur la roue

Effectivement, la TEI définit des éléments qui ressemblent beaucoup aux vôtres. Pourquoi pas s'en servir?

```
<schemaSpec
  source="/usr/share/xml/tei/odd/Source/Guidelines/en/guidelines-en.xml"
  start="div"
  ident="simpleS-2">
  <elementRef key="div"/>
  <elementRef key="p"/>
  <elementRef key="ptr"/>
</schemaSpec>
```

(L'attribut @source indique où trouver les définitions standardisées des éléments qu'on souhaite inclure)

Pourquoi se servir des définitions existantes?

- Principe du moindre effort
- Vos ressources sont maintenant dotées d'une sémantique plus ou moins standardisée
- (Mais vos propres interprétations restent disponibles dans votre propre documentation)

Et rien n'empêche le "mix and match" :

```
<schemaSpec
  source="/usr/share/xml/tei/odd/Source/Guidelines/en/guidelines-en.xml"
  start="stuff"
  ident="simpleS-3">
  <elementSpec ns="" ident="stuff">
    <desc>élément racine d'un schéma simplissime</desc>
    <desc xml:lang="en">Root element for a very simple schema</desc>
    <content>
<!-- as before -->
    </content>
  </elementSpec>
  <elementRef key="p"/>
  <elementRef key="ptr"/>
</schemaSpec>
```


Dans le monde réel, les éléments se déplacent en groupes

Un module est une collection d'éléments nommée.. La TEI en fournit 22. Pour inclure l'une de ces collections, il suffit de la faire référence avec l'élément `<moduleRef>` :

```
<schemaSpec start="TEI" ident="testschéma-4">  
  <moduleRef key="core"/>  
  <moduleRef key="header"/>  
  <moduleRef key="textstructure"/>  
</schemaSpec>
```

Tout élément TEI appartient à un seul module et possède un nom unique.

Recap

- La TEI propose plusieurs modules
- Chaque module propose plusieurs spécifications d'élément
- Chaque spécification contient
 - un nom canonique (`<gi>`)
 - (facultativement) des noms équivalents en plusieurs langues
 - une description de sa fonction (également disponible en plusieurs langues)
 - une indication des classes auxquels il appartient
 - une définition pour chacun de ses attributs
 - une de son contenu (content model)
 - exemples d'usage ; notes
- une spécification schéma (`<schemaSpec>`) TEI contient
 - des références aux modules et/ou aux éléments
 - des (re)déclarations d'élément, classe, ou macro
- Un document TEI document contenant une spécification de schéma s'appelle un ODD (One Document Does it all)

Liste des TEI modules

analysis	Simple analytic mechanisms
certainty	Certainty and uncertainty
core	éléments common to all TEI documents
corpus	Header extensions for corpus texts
declarefs	Feature system déclarations
dictionaries	Printed dictionaries
drama	Performance texts
figures	Tables, formulae, and figures
gaiji	Character and glyph documentation
header	The TEI Header
iso-fs	Feature structures
linking	Linking, segmentation and alignment
msdescription	Manuscript Description
namesdates	Names and dates
nets	Graphs, networks and trees
spoken	Transcribed Speech
tagdocs	Documentation of TEI modules
tei	déclarations for datatypes, classes, and macros available to all TEI modules
textcrit	Text criticism
textstructure	Default text structure
transcr	Transcription of primary sources
verse	Verse structures

Choisir selectivement (1)

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

```
<schemaSpec start="TEI" ident="testschéma-4a">  
  <moduleRef key="core" except="mentioned quote said"/>  
  <moduleRef key="header"/>  
  <moduleRef key="textstructure"/>  
</schemaSpec>
```

ou également :

```
<schemaSpec start="TEI" ident="testschéma-4b">  
  <moduleRef key="core"/>  
  <moduleRef key="header"/>  
  <moduleRef key="textstructure"/>  
  <elementSpec ident="mentioned" mode="delete"/>  
  <elementSpec ident="quote" mode="delete"/>  
  <elementSpec ident="said" mode="delete"/>  
</schemaSpec>
```

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

Choisir selectivement (2)

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

```
<schemaSpec start="TEI" ident="testschéma-4b">  
  <moduleRef key="core"/>  
  <moduleRef key="header"/>  
  <moduleRef key="textstructure" include="body div"/>  
</schemaSpec>
```

ou également :

```
<schemaSpec start="TEI" ident="testschéma-4b">  
  <moduleRef key="core"/>  
  <moduleRef key="header"/>  
  <elementRef key="div"/>  
  <elementRef key="body"/>  
</schemaSpec>
```

Unifications de déclarations multiples

Comme noté ci-dessus , l'attribut @mode sert à contrôler les actions d'un processeur ODD qui découvre plusieurs déclarations pour un seul composant.

valeur de @mode	au premier rencontre	au rencontre suivant
add	ajouter déclaration à la schéma; traiter enfants dans mode add	signaler erreur
replace	signaler erreur	ajouter déclaration; traiter nouveaux enfants en mode replace; supprimer enfants existants
change	signaler erreur	traiter enfants identifiables (i.e. portant un identifiant) selon leur mode; traiter enfants non-identifiables en mode replace; retenir par défaut enfants existants
delete	signaler erreur	supprimer déclaration existante, et ses enfants

éléments "identifiables"

Les éléments '*-spec' sont tous membres d'une classe `att.identifiable` qui les fournit un attribut `@ident` équivalent à l'attribut global `@xml:id`; on s'en sert pour identifier la déclaration. Pour faire référence à une telle déclaration, nous nous servons du `@key` attribut:

```
<elementRef key="bar"/>  
<!-- implique l'existence ailleurs de ... -->  
<elementSpec ident="bar">  
<!-- .... -->  
</elementSpec>
```

Similarly:

```
<moduleRef key="foo"/>  
<!-- implique l'existence ailleurs de ... -->  
<moduleSpec ident="foo"/>
```

Pour les attributs et les valeurs, `<attDef>` et `<valItem>` sont pareilles.

Specification des listes de valeurs

Les valeurs legales d'un attribut peuvent etre specifiés par un `<datatype>` et/ou par `<valList>`.

Un besoin assez commun est de specifier une enumeration (une liste -- ouverte ou fermée -- des valeurs)

```
<attDef ident="status">
  <desc>indique l'etat courant du système selon un code de couleur</desc>
  <defaultVal>green</defaultVal>
  <valList type="closed">
    <valItem ident="red">
      <desc>fermeture du système complete</desc>
    </valItem>
    <valItem ident="orange">
      <desc>fermeture du système imminent</desc>
    </valItem>
    <valItem ident="green">
      <desc>etat du système normal</desc>
    </valItem>
    <valItem ident="white">
      <desc>etat du système inconnu</desc>
    </valItem>
  </valList>
</attDef>
```


Datatypes

Servent à contraindre les valeurs legales d'un attribut :

```
<attDef ident="status">
  <datatype>
    <rng:ref name="data.enumerated"/>
  </datatype>
<!-- ... implique l'existence d'un vallist -->
</attDef>
<attDef ident="lastUpdated">
  <datatype>
    <rng:ref name="data.temporalExpr.w3c"/>
  </datatype>
</attDef>
```

Les datatypes définis par TEI sont, pour la plupart, mappés sur des definitions W3C, en se servant d'un `<macroSpec>`

Specification des patron s

L'élément `<macroSpec>` sert à associer un nom et une chaîne de caractères. On s'en sert typiquement pour

- la définition de quelques modèles de contenu très répandus
- la définition des datatypes

```
<macroSpec ident="macro.foo">
  <desc>a content model i plan to reuse often</desc>
  <content>
<!-- patron RELAXNG definissant la modele -->
  </content>
</macroSpec>
<macroSpec ident="data.foo">
  <desc>a new datatype i just invented</desc>
  <content>
<!-- RELAXNG pattern defining the datatype -->
  </content>
</macroSpec>
```

Constraints schématron

- Une spécification d'élément peut aussi contenir un élément `<constraintSpec>` élément rassemblant des règles exprimées en ISO schématron

un cartoon doit inclure une graphie

Attention :

- traitement de ces règles nécessitent un étape additionnel dans la validation des documents
- ces règles ne sont pas forcément traitable par toute langue de schéma

Le système de classe de la TEI: mode d'emploi

En définissant un nouveau élément on a besoin de considérer :

- son nom, sa description, etc.
- ses attributs
- son contenu
- sa contexte

Le système de classe de la TEI facilite toutes ses considerations (sauf la première).

Les classes attributives ("Attribute Classes")

- Ces classes portent des noms commençant par **att.**; ex. `att.naming`, `att.typed`
- Chacun de ces classes contient un ensemble de déclarations d'attributs, qui sont hérités par les membres de la classe. Par ex, tous les membres de `att.naming` en héritent les attributs `@key` et `@ref`; tous les membres de classe `att.typed` en héritent `@type` et `@subtype` et ainsi de suite
- Pour qu'un élément porte l'attribut `@type`, donc, nous ajoutons cet élément à la classe `att.typed`.
- (Il reste possible de définir des attributs "locaux" avec l'élément indépendamment, bien sur)

Les classes modelisantes (Model Classes)

- Ces classes portent des noms qui commencent en "model." et rassemblent des éléments qui sont permis à la même position dans l'arborescence défini par un schéma. On peut définir cette position en deux manières:
 - si le nom de la classe contient le mot **Like**, par exemple **model.biblLike**, ses membres peuvent apparaître aux mêmes contextes que l'élément indiqué (ici, **<bibl>**)
 - si le nom de la classe contient le mot **Part**, par exemple **model.biblPart**, ses membres peuvent apparaître à l'intérieur de l'élément indiqué
- En l'occurrence, les classes sont souvent sous-divisées. Par ex, **model.pPart** (les éléments qui peuvent apparaître à l'intérieur d'un **<p>**) comporte:
 - **model.pPart.edit** éléments représentant les interventions éditoriales par ex **<corr>**, **** etc.
 - **model.pPart.data** 'data-like' éléments par ex **<name>**, **<num>**, **<date>** etc.
 - **model.pPart.msdesc** des éléments sous-paragraphe qui sont spécifiques aux descriptions des manuscrits par ex **<seal>** ou **<origPlace>**

Specification des classes

- L'élément `<classSpec>` sert à déclarer une classe. Son attribut `@type` indique s'il s'agit d'une classe attributive (`type="att"`) ou modelisante (`type="model"`)
- Le `<classSpec>` pour une classe modelisante ne contient que sa description et son identifiant; celui d'une classe attributive contient aussi un `<attList>`

```
<classSpec ident="model.foo" type="model">  
  <desc>classe contenant tous les éléments avec noms idiots, inventés par raison  
    pédagogique </desc>  
</classSpec>
```

```
<classSpec ident="att.foo" type="atts">  
  <desc>classe fournissant l'attribut <att>bar</att>  
  </desc>  
  <attList>  
    <attDef ident="bar">  
      <!-- ... -->  
    </attDef>  
  </attList>  
</classSpec>
```

Selection des classes

- Un élément indique les classes auxquelles il appartient en se servant d'un ou plusieurs `<memberOf>` éléments, regroupés dans un élément `<classes>` dans son `<elementSpec>`.
- Une classe indique les classes dont elle est sous-classe de la même manière.

```
<elementSpec ident="bidule">
  <desc>élément nouveau inventé purement par motivation pédagogique</desc>
  <classes>
    <memberOf key="model.foo"/>
    <memberOf key="att.foo"/>
  </classes>
<!-- ... -->
</elementSpec>
```