Schémas XML

Ingénierie Documentaire
http://doc.crzt.fr

Stéphane Crozat

14 septembre 2016

Table des matières

Introduction aux schémas XML

I

1. Notion de document valide

Définition : Schéma

Un schéma est une description de la structure que doit respecter un document lui faisant référence, c'est à dire qu'il établit la liste des éléments XML autorisés (avec leurs attributs), ainsi que l'agencement possible de ces éléments.

On parle aussi de *grammaire*, au sens où le schéma définit l'enchaînement autorisé des balises et vient en *complément de la syntaxe XML* (qui elle est indépendante d'un schéma particulier).

Définition : Document valide

Un document XML * bien formé est dit valide pour un schéma donné s'il respecte les règles structurelles imposées par ce schéma.

Ce contrôle de la structure permet :

- De s'assurer l'homogénéité structurelle des documents de même type.
- Le traitement automatique d'un ensemble de documents de même type (mise en forme, stockage, extraction d'informations...).
- La création de formats standard et leur respect.

Exemple : Exemples de langages de schéma

Il existe plusieurs langages de définition schéma, mais les trois principaux sont :

- Document Type Définition (W3C) : Un langage hérité de SGML qui fait partie du standard XML
- W3C XML Schema (W3C) : Une alternative aux DTD destiné à moderniser et compléter ce langage historique
- Relax NG (OASIS, ISO) : Une autre alternative, compromis entre W3C XML Schema et DTD

2. Document Type Definition

Le formalisme de définition de schéma DTD est le premier qui a été introduit dès la première version du standard XML. Il est en fait intégré au standard W3C de XML.

Il est directement hérité de la norme SGML.

Les DTDs utilisent un langage spécifique (non XML) pour définir les règles structurelles. Un fichier de DTD peut contenir principalement deux types de déclarations :

- *des déclarations d'éléments,*
  indiquent les éléments pouvant être inclus dans un document et l'organisation du contenu de chaque élément (éléments fils ou texte).
- *des déclarations d'attributs,*
  définissent les attributs pouvant être associés à un élément ainsi que leur type.

Exemple : Exemple de DTD

```
<!ELEMENT document (paragraphe+)>
<!ATTLIST document type CDATA #REQUIRED>
<!ELEMENT paragraphe (#PCDATA)>
```

Exemple : Exemple de document XML valide

```
<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE document SYSTEM "document.dtd">
<document type='memo'>
  <paragraphe>Lorem ipsum dolor sit amet.</paragraphe>
  <paragraphe>Consectetur adipiscing elit.</paragraphe>
  <paragraphe>Sed do eiusmod tempor.</paragraphe>
</document>

```

3. W3C XML Schema

Les XML Schema ont été proposés par le W3C pour permettre de dépasser les limites des DTD.

http://www.w3.org/XML/Schema

On notera en particulier :

- une syntaxe XML
- l'extension de l'expression des règles d'organisation structurelle (héritage, réutilisation, etc.)
- l'ajout d'un langage de typage des éléments (particulièrement utile pour les format XML orientés données)

Exemple : Exemple de DTD

```
<!ELEMENT document (paragraphe+)>
<!ATTLIST document type CDATA #REQUIRED>
<!ELEMENT paragraphe (#PCDATA)>
```

Exemple : Exemple de W3C XML Schema correspondant

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="document">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="paragraphe"/>
            </xs:sequence>
            <xs:attribute name="type" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="paragraphe" type="xs:string"/>
</xs:schema>
```

4. Regular Language for XML Next Generation

RelaxNG (REgular LAnguage for XML Next Generation) est un langage de schéma XML.

- RelaxNG est une alternative aux DTD et à W3C XML Schema, qui combine les avantages de ces deux autres langages.
- RelaxNG est un standard OASIS et une norme ISO/CEI.
- Deux syntaxes : une syntaxe XML (alternative à W3C Schema) et une syntaxe compacte (alternative aux DTD).
- RelaxNG ne définit que la structure (comme les DTD) et utilise W3C XML Schema pour le typage des données.

http://relaxng.org/

Complément

Le standard est porté par James Clark depuis ses travaux sur Trex (il est issu de la fusion de Trex et Relax de Murata Makoto).

Exemple : Exemple de schémas publics définis en Relax NG

- OpenDocument (format bureautique)
- DocBook (format documentaire)
- Atom (syndication)

Exemple : Exemple de DTD

```
<!ELEMENT document (paragraphe+)>
<!ATTLIST document type CDATA #REQUIRED>
```

```
3 <!ELEMENT paragraphe (#PCDATA)>
```

Exemple : Exemple de schéma RelaxNG correspondant (syntaxe XML)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <grammar xmlns="http://relaxng.org/ns/structure/1.0">
3     <start>
4         <element name="document">
5             <attribute name="type"/>
6             <oneOrMore>
7                 <element name="paragraphe">
8                     <text/>
9                 </element>
10                </oneOrMore>
11        </element>
12    </start>
13 </grammar>
```

Exemple : Exemple de schéma RelaxNG correspondant (syntaxe compacte)

```
1 start = element document {
2     attribute type {text},
3     element paragraphe {text}+
4 }
```

Exemple : Autre exemple de schéma RelaxNG correspondant (syntaxe XML, patterns nommés)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <grammar xmlns="http://relaxng.org/ns/structure/1.0">
3     <start>
4         <ref name="document"/>
5     </start>
6     <define name="document">
7         <element name="document">
8             <attribute name="type"/>
9             <oneOrMore>
10                <ref name="paragraphe"/>
11            </oneOrMore>
12        </element>
13    </define>
14    <define name="paragraphe">
15        <element name="paragraphe">
16            <text/>
17        </element>
18    </define>
19 </grammar>
```

Exemple : Autre exemple de schéma RelaxNG correspondant (syntaxe compacte, patterns nommés)

```
1 start = document
2 document = element document {attribute type {text}, paragraphe+ }
3 paragraphe = element paragraphe {text}
```

Définition de type de document

II

1. Déclaration d'éléments

Syntaxe : Déclaration d'éléments

Les déclarations d'éléments sont de la forme :

```
<!ELEMENT nom (modèle)>
```

où :

- Le `nom` obéit aux mêmes règles que les noms dans les balises, c'est à dire commençant par un caractère alphabétique (ou un tiret bas) suivi des caractères alphanumériques, point ou tiret bas.
- Le `modèle` décrit la combinaison d'éléments et de texte qui pourra être contenue dans cet élément :
  - `(nom d'un ou plusieurs élément fils)`
    Indique que l'élément peut contenir un ou des fils ayant le nom indiqué.
  - `(#PCDATA)`
    Indique la possibilité de contenir un flot de caractères (*Parsed Character Data*).

Exemple : Exemple de déclaration d'éléments

```
<!ELEMENT texte (paragraphe)>
<!ELEMENT paragraphe (#PCDATA)>
```

Un texte contient un paragraphe qui contient un flux de caractères.

```
<texte>
    <paragraphe>Ceci est un flux de caractères</paragraphe>
</texte>
```

Syntaxe : Séparateur de structuration

Les éléments fils déclarés peuvent être combinés pour décrire en détail la structure du contenu de l'élément en les séparant par :

- `,`
  Indique une relation d'ordre (connecteur logique `AND` avec une notion d'ordre) : si par exemple si le modèle est `(x,y)` alors l'élément x devra être présent, et ce avant l'élément y.

- |
  Indique une alternative (connecteur logique "ou exclusif" XOR). Le modèle (x|y) indique "soit x soit y".

L'utilisation de parenthèses permet de créer des sous-listes dans la liste principale pour lesquelles les suffixes de cardinalité sont également applicables.

Exemple : Exemple de déclaration de deux éléments fils ordonnés

```
<!ELEMENT texte (titre, paragraphe)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT paragraphe (#PCDATA)>
```

```
<texte>
    <titre>Ceci est le flux de caractères du titre</titre>
    <paragraphe>Ceci est le flux de caractères du paragraphe</paragraphe>
</texte>
```

Exemple : Exemple de déclaration d'alternative

```
<!ELEMENT texte ((titre | accroche), paragraphe)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT accroche (#PCDATA)>
<!ELEMENT paragraphe (#PCDATA)>
```

Syntaxe : Suffixes de cardinalité

Les éléments fils peuvent être déclarés avec des suffixes permettant d'exprimer leur cardinalité :

- ? : l'élément devra être présent de 0 à 1 fois.
- * : l'élément devra être présent de 0 à n fois.
- + : l'élément devra être présent de 1 à n fois.

L'absence de suffixe indique que l'élément doit apparaître une et une seule fois.

Exemple : Exemple général

Les éléments x, y, z1 et z2 sont représentés vides pour alléger l'exemple.

```
<!ELEMENT mon_elem (x?, y*, (z1, z2)+)>
```

permet :

```
<mon_elem> <x/><y/><z1/><z2/> </mon_elem>
<mon_elem> <z1/><z2/> </mon_elem>
<mon_elem> <z1/><z2/><z1/><z2/> </mon_elem>
<mon_elem> <x/><y/><y/><y/><z1/><z2/><z1/><z2/> </mon_elem>
<mon_elem> <y/><y/><y/><y/><z1/><z2/> </mon_elem>
...
```

Attention : Élément racine

L'élément racine n'est pas spécifié dans une DTD, il le sera dans la référence à la DTD faite depuis le fichier XML. Cela permet en particulier à une DTD de spécifier plusieurs langages (plusieurs éléments racines), même si cela n'est pas en général conseillé.

Par convention on déclarera en premier l'élément racine.

2. Déclaration d'éléments EMPTY et ANY

Syntaxe : Element de type ANY

```
1 <!ELEMENT nom_element ANY>
```

L'élément pourra contenir des flots de caractères ainsi que n'importe quels éléments déclarés par ailleurs.

Syntaxe : Element de type EMPTY

Il est possible de déclarer un élément vide en utilisant la syntaxe EMPTY :

```
1 <!ELEMENT nom_element EMPTY>
```

L'élément doit être vide. Il contiendra néanmoins généralement des attributs.

Remarque

ANY et EMPTY ne sont pas combinables avec d'autres définitions d'éléments fils, mais sont compatibles avec des définitions d'attributs.

Exemple : Exemple 1 : Le paramètre ANY

```
1 <!ELEMENT monElement1 ANY>
2 <!ELEMENT monElement2 EMPTY>
3 permet :
4    <monElement1><monElement2/>bonjour</monElement1>
5    <monElement1>au revoir<monElement2/></monElement1>
6    <monElement1></monElement1>
7    ...
```

Exemple : Exemple 2 : Le paramètre EMPTY

```
1 <!ELEMENT monElement1 EMPTY>
2 permet :
3 <monElement1/> ou <monElement1></monElement1>
```

3. Déclarations de listes d'attributs

Syntaxe : Déclaration d'attribut

Les déclarations d'attributs correspondent à la forme générale :

```
<!ATTLIST nom-élément nom-attribut type-attribut déclaration-de-contrainte>
```

où :

- *nom-élément :*
  Nom de l'élément XML * pour lequel les attributs déclarés seront applicables.
- *nom-attribut :*
  Nom de l'attribut, des éléments différents peuvent avoir des attributs de même nom sans qu'il y ait de confusion possible car un attribut est toujours déclaré en même temps que l'élément auquel il est attaché.
- *type-attribut :*
  Les deux types principaux sont :
  - *CDATA*
    L'attribut aura pour valeur une chaîne de caractères.
  - *Liste de choix*
    Une liste de noms symboliques correspondant aux valeurs possibles pour l'attribut et se présentant sous la forme : `(choix1 | choix2 | ... | choixN)`.
- *déclaration-de-contrainte :*
  Les deux formes principales de contrainte sont :
  - *#REQUIRED*
    L'attribut est obligatoire.
  - *#IMPLIED*
    L'attribut est facultatif.

Remarque : Déclaration de liste d'attributs

La syntaxe `ATTLIST` permet de déclarer des listes d'attributs, car il est possible de répéter le *pattern* : `nom-attribut type-attribut déclaration-de-contrainte`.

Exemple : Déclaration de liste d'attributs

```
<!ELEMENT x EMPTY>
<!ATTLIST x
  att1 CDATA #REQUIRED
  att2 (a | b | c) #IMPLIED
>
```

Complément : Autres types d'attributs

En plus des types `CDATA` et "liste de choix", les attributs peuvent avoir les types suivants :

- *ID ou IDREF*

Sont utilisés pour définir des liens à l'intérieur d'un document, ID identifiant de manière unique tous les éléments pouvant être référencés et IDREF indiquant la référence à cet identifiant.

- *NMTOKEN ou NMTOKENS*
  Permet à l'attribut de prendre pour valeur un ou des noms symboliques quelconques, formés de caractères alphanumériques.
- *ENTITY ou ENTITIES*
  L'attribut prendra pour valeur le nom d'une ou plusieurs entités externes non XML (images...).
- *NOTATION*
  Est utilisé pour des éléments ayant un contenu non XML, l'attribut aura alors pour valeur le nom de l'application qui a été associée à l'application externe traitant le type de contenu concerné lors d'une déclaration de type NOTATION.

Complément : Autres déclarations de contraintes

En plus des déclarations #IMPLIED et #REQUIRED, les attributs peuvent avoir les déclaration de contrainte :

- *Valeur par défaut*
  Indique la valeur par défaut prise par l'attribut (en accord avec son type) s'il n'est pas renseigné. L'attribut peut donc être renseigné ou non (comme un #IMPLIED), mais s'il ne l'est pas il prend la valeur par défaut spécifiée. Se note 'valeur'.
- *#FIXED 'valeur'*
  L'attribut prend toujours la valeur indiquée, il est "constant". Il doit donc toujours être renseigné (comme un #REQUIRED), mais toujours avec la même valeur.

Complément : Voir aussi

ID et REFID (cf. p.12)

4. Identifiants et références (ID et IDREF)

Définition : Élément identifié par un attribut identifiant (ID)

Les attributs de type ID permettent d'identifier de façon unique un élément. L*a valeur d'un attribut de type ID est unique parmi toutes les valeurs des attributs ID de tout le document* (sinon le document n'est pas valide, principe d'unicité).

On notera également que :

- la valeur d'un attribut ID est de type NMTOKEN
- un élément ne peut avoir qu'un seul attribut de type ID
- un attribut ID peut être optionnel (#IMPLIED) ou obligatoire (#REQUIRED)

Syntaxe

```
1 <!ELEMENT e (...)>
2 <!ATTLIST e
```

```
3 a ID #REQUIRED>
```

Définition : Réference à des éléments identifiés (IDREF)

Les attributs de type IDREF permettent de faire référence à un élément identifié, par la valeur de l'attribut ID correspondant. *La valeur d'un attribut IDREF doit correspondre à la valeur d'un attribut ID dans le document* (sinon le document n'est pas valide, principe d'intégrité référentielle).

Syntaxe

```
1 <!ELEMENT e (...)>
2 <!ATTLIST e
3 a IDREF #REQUIRED>
```

Remarque : IDREFS

Un attribut de type IDREFS permet de faire référence à *plusieurs* élément identifiés, le séparateur étant un espace (référence de cardinalité N:M).

Exemple : DTD avec ID, IDREF, IDREFS

```
1 <!-- dtdid.dtd -->
2 <!ELEMENT e1 (e2 | e3 | e4)*>
3 <!ELEMENT e2 (#PCDATA)>
4 <!ELEMENT e3 (#PCDATA)>
5 <!ELEMENT e4 (#PCDATA)>
6 <!ATTLIST e2
7 id ID #REQUIRED>
8 <!ATTLIST e3
9 ref IDREF #IMPLIED>
10 <!ATTLIST e4
11 refs IDREFS #IMPLIED>
```

Exemple : Extrait de document XML valide par rapport à dtdid.dtd

```
1 <e1>
2   <e2 id="id1">...</e2>
3   <e2 id="id2">...</e2>
4   ...
5   <e3 ref="id1"/>
6    ...
7   <e4 refs="id1 id2"/>
8 </e1>
```

5. Contenu mixte (mixed content)

Définition

On appelle *mixed content* (contenu mixte en français) un élément XML contenant à la fois du flux texte (#PCDATA) et des éléments fils.

On appelle élément *inline* un tel élément, au sens d'inclus dans le flux de caractères : "dans la ligne".

Syntaxe

```
1 <!ELEMENT mixedElement (#PCDATA | inlineElement1 | inlineElement2 )*>
```

Remarque

`#PCDATA` doit être déclaré en premier dans la liste.

Attention

Seule la forme `(#PCDATA | ...)*` est autorisée :

- `(#PCDATA | ...)+` induirait une obligation non contrôlable, car #PCDATA inclus la chaîne vide.
- `(#PCDATA | ...)` ou `(#PCDATA | ...)+` induiraient une alternative ou un ordre non contrôlables, car on ne peut pas différencier les caractères non signifiants entre des éléments XML et les caractères signifiants faisant partie du flux texte.
- ...

6. Déclaration de DTD dans les fichiers XML

Syntaxe

L'instruction DOCTYPE situé en début de fichier XML (entre l'entête et la balise racine) permet d'associer un fichier XML à la DTD qui permet de la valider. Les parseurs validant pourront ainsi directement informer sur la validité du fichier.

```
1 <?xml version="1.0"?>
2 <!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "http://docs.oasis-open.org/dita/v1.
  1/OS/dtd/topic.dtd">
3 <topic>
4 ...
```

Syntaxe : Déclaration de DTD privée

Une DTD privée est une DTD réalisée pour ses besoins propres (langage local).

```
1 <!DOCTYPE rootElement SYSTEM "url">
```

avec :

- `rootElement` le nom de l'élément racine
- `url` est une URL décrivant le chemin de la DTD : en local sur un disque, en chemin absolu ou relatif par rapport à la position du fichier XML, ou sur Internet typiquement.

Exemple : Déclaration de DTD privées

- <!DOCTYPE topic SYSTEM "topic.dtd">
- <!DOCTYPE topic SYSTEM "/home/stc/topic.dtd">
- <!DOCTYPE topic SYSTEM "http://www.utc.fr/ics/~stc/topic.dtd">

Syntaxe : Déclaration de DTD publique

Une DTD publique est une DTD standard mise à disposition pour une communauté élargie.

```
<!DOCTYPE rootElement SYSTEM "publicName" "url">
```

avec :

- `publicName` est un nom public pour la DTD, qui permettra à un système informatique de rechercher en priorité la DTD via un catalogue local.
- `url` est une URL décrivant le chemin de la DTD comme pour le mode SYSTEM.

Exemple : Les DTD DITA et XHTML

- <!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "http://docs.oasis-open.org/dita/v1.1/OS/dtd/topic.dtd">
- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

7. Exemple de DTD

Exemple : DTD (inspirée de DITA)

```
<!-- topic-like.dtd -->
<!ELEMENT topic (title,body)>
<!ATTLIST topic
id NMTOKEN #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT body (p)+>
<!ELEMENT p (#PCDATA|term)*>
<!ELEMENT term (#PCDATA)>
```

Exemple : Contenu valide par rapport à topic-like.dtd

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE topic SYSTEM "topic-like.dtd">
<topic id="topic-1">
<title>Topic title</title>
<body>
<p>This is un exemple of DITA <term>topic</term>.</p>
<p>...</p>
</body>
</topic>
```

8. Inconvénients et avantages des DTD

Les DTD * présentent des inconvénients et des limites technologiques qui ont suscité le développement de certaines solutions concurrentes.

Le premier défaut des DTD * est qu'elles s'écrivent avec un langage spécifique, avec sa propre syntaxe, ce qui est d'autant plus déroutant que l'on traite avec un méta-langage (permettant de créer des langages !). On attendrait donc des DTD qu'elles s'expriment en XML.

De plus, avec l'essor du XML * et son utilisation conjointe avec d'autres applications (systèmes de base de données notamment), le couple XML+DTD ne fournissait pas une solution satisfaisante car :

- Il ne garantit pas que les éléments terminaux (#PCDATA) contiennent des données d'un certain type ou dans un format contrôlable
- Il ne permet pas dans le cas général d'exprimer exactement la cardinalité d'un élément (indiquer qu'un tableau doit contenir un nombre x de colonnes et y de lignes).

Exemple : Langages concurrents des DTD

- XML Schema (W3C)
- Relax NG (OASIS)
- Schematron
- Examplotron
- etc.

Attention : Quel utilisation des DTD ?

Les DTD restent aujourd'hui très utilisées, malgré leurs lacunes, pour leur simplicité et leur efficacité dans la description des langages de description documentaires.

Elles ne sont en revanche plus utilisées pour les langages orientés données, où on leur préfère largement les XML Schema par exemple.

Exemple : Une DTD standard

La DocBook (*www.docbook.org*) est une DTD posant les règles de structuration de livres techniques (en particulier dans le domaine de l'informatique) en chapitres, parties, etc

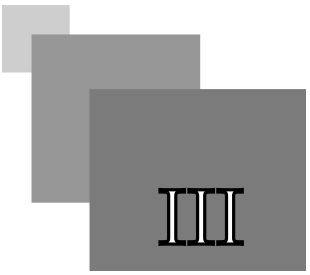
9. La syntaxe DTD en résumé

La syntaxe des DTD repose sur un langage déclaratif, permettant de spécifier les éléments autorisés et pour chaque élément autorisé les fils qu'il peut contenir. Les aspects essentiels de la syntaxe sont présentés dans le tableau ci-dessous.

Aspects	Syntaxe	Explications
Elément racine	<!ELEMENT a (...)>	L'élément racine est déclaré en premier par soucis de lisibilité (ce n'est pas obligatoire), c'est le seul élément à n'être inclus dans aucun autre.
Elément contenant d'autres éléments	<!ELEMENT a (b)>	Tout élément est déclaré en indiquant les fils qui lui sont autorisés (ici on déclare l'élément a qui contient un fils b)
Le 'ET' (,)	<!ELEMENT a (b,c)>	La virgule permet d'indiquer la relation ET entre les élément fils (ici a contient un b ET un c).
Le 'OU' (|)	<!ELEMENT a (b|c)>	La barre permet d'indiquer la relation OU entre les élément fils (ici a contient un b OU un c).
Le '1' (par défaut)	<!ELEMENT a (b)>	Par défaut tout élément fils déclaré doit être présent une et une seule fois dans le père (ici a contient obligatoirement un et un seul b)
Le '0 ou 1' (?)	<!ELEMENT a (b?)>	Le point d'interrogation permet de spécifier que le fils est optionnel (ici a contient zéro ou un b).
Le '0 à N' (*)	<!ELEMENT a (b*)>	L'étoile permet de spécifier que le fils est optionnel et que le père peut contenir plusieurs fois le fils (ici a contient zéro, un ou plusieurs b).
Le '1 à N' (+)	<!ELEMENT a (b+)>	Le plus permet de spécifier que le père contient au moins une fois le fils, mais peut le contenir plusieurs fois (ici a contient un ou plusieurs b).
Attribut d'un élément	<!ATTLIST NomElement NomAttribut Type Contrainte>	Lorsqu'un attribut est déclaré on spécifie le nom de l'élément qu'il concerne, son nom, son type (libre, énuméré, identifiant unique, etc.) et la contrainte qui lui est asociée (obligatoire, optionnel ou fixé)
Elément vide	<!ELEMENT a EMPTY>	Un élément vide est spécifié à l'aide du mot clé EMPTY à la place de ses fils.
Elément contenant du contenu	<!ELEMENT a (#PCDATA)>	Pour spécifier qu'un élément contient du contenu, on utilise le mot clé #PCDATA à la place du nom d'un élément fils (#PCDATA équivaut à chaîne de caractères).
Commentaires	<!-- commentaires -->	On peut ajouter des commentaires en respectant la même syntaxe qu'en XML

Aspects principaux de la syntaxe des DTD

RelaxNG



Spécifications : *http://www.oasis-open.org/committees/relax-ng/spec.html*

1. Relax NG : Syntaxe XML

Syntaxe : Syntaxe générale

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="...">
      ...
    </element>
  </start>
</grammar>
```

Syntaxe : Éléments

```
<element name="">
  <element name="">
    ...
  </element>
</element>
```

Syntaxe : Attributs

```
<element name="">
  <attribute name="">
    ...
</element>
```

Syntaxe : Nœuds texte

```
<element name="">
  <text/>
</element>
```

Syntaxe : Cardinalité

```
<element name="">
```

```
  <zeroOrMore>
    <element name="">
  </zeroOrMore>
  <oneOrmore>
    <element name="">
  </oneOrMore>
...
```

Syntaxe : Optionalité

```
<element name="">
  <optional>
    <element name="">
  </optional>
...
```

Syntaxe : Alternative

```
<element name="">
  <choice>
    <element name="">
    <element name="">
    <element name="">
  </choice>
```

Syntaxe : Énumération

```
<element name="">
  <choice>
    <value></value>
    <value></value>
    <value></value>
  </choice>
```

```
<attribute name="">
  <choice>
    <value></value>
    <value></value>
    <value></value>
  </choice>
```

Complément

http://www.oasis-open.org/committees/relax-ng/tutorial.html

http://xmlfr.org/oasis/committees/relax-ng/tutorial-20011203-fr.html

2. Patterns nommés

Syntaxe : Déclaration de pattern

```
<define name="">
...
```

```
3 </define>
```

Syntaxe : Appel de pattern

```
1 <ref name=""/>
```

Syntaxe : Syntaxe générale avec pattern nommés

```
1 <grammar xmlns="http://relaxng.org/ns/structure/1.0">
2   <start>
3     <element name="">
4       <ref name=""/>
5     </element>
6   </start>
7   <define name="">
8 ...
9   <define name="">
10 ...
11 </grammar>
```

Attention : Un pattern n'implique pas un élément

Dans l'exemple ci-après, il est nécessaire de déclarer l'élément `paragraphe` dans le pattern `Paragraphe`.

```
1 <grammar xmlns="http://relaxng.org/ns/structure/1.0">
2 <start>
3   <element name="texte">
4     <zeroOrMore>
5       <ref name="Paragraphe"/>
6     </zeroOrMore>
7   </element>
8 </start>
9 <define name="Paragraphe">
10   <element name="paragraphe">
11     <text/>
12   </element>
13 </define>
14 </grammar>
```

Conseil : Convention

On adoptera la convention minimale suivante : si tous les éléments du langage XML commencent par une majuscule, les patterns commenceront par une majuscule.

Si certains éléments commencent par une majuscule ou si l'on souhaite limiter les risques de confusion, on pourra adopter une convention plus explicite, comme préfixer les patterns par `p_` par exemple.

3. Ensembles non ordonnés

Définition : Intercalation

Le pattern `interleave` permet de définir des ensembles non ordonnés.

Syntaxe

```
<element name="...">
  <interleave>
    <element name="...">...</element>
    <element name="...">...</element>
    ...
  </interleave>
</element>
```

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
<element name="paragraph">
  <interleave>
    <element name="a"><text/></element>
    <element name="b"><text/></element>
  </interleave>
</element>
</start>
</grammar>
```

Autorise :

- `<paragraph><b/><a/></paragraph>`
- `<paragraph><a/><b/></paragraph>`

Complément

http://relaxng.org/tutorial-20011203.html#IDAN1YR

4. Contenu mixte

Rappel

Contenu mixte (mixed content) (cf. p.13)

Syntaxe

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
  <element name="mixedElement">
    <zeroOrMore>
```

```
    <choice>
      <text/>
      <element name="inlineElement">...</element>
      ...
    </choice>
  </zeroOrMore>
 </element>
</start>
</grammar>
```

Exemple

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
 <element name="paragraph">
  <zeroOrMore>
    <choice>
      <text/>
      <element name="emphasis"><text/></element>
      <element name="foreign"><text/></element>
    </choice>
  </zeroOrMore>
 </element>
</start>
</grammar>
```

Syntaxe : Mixed content avec interleave

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
<element name="mixedElement">
 <interleave>
  <text/>
  <zeroOrMore>
    <element name="inlineElement">...</element>
   </zeroOrMore>
  <zeroOrMore>
    ...
   </zeroOrMore>
  ...
 </interleave>
</element>
</start>
</grammar>
```

Exemple : Mixed content avec interleave

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
<element name="paragraph">
 <interleave>
  <text/>
  <zeroOrMore>
  <element name="emphasis"><text/></element>
    </zeroOrMore>
```

```
    <zeroOrMore>
      <element name="foreign"><text/></element>
    </zeroOrMore>
  </interleave>
</element>
</start>
</grammar>
```

5. Namespace cible

Syntaxe : Namespace cible (unique pour tout le schéma)

```
<grammar ns="">
```

Syntaxe : Namespace cible (pour un élément particulier)

```
<element name="" ns="">
```

6. Inclusion de schémas

Syntaxe : Inclusion simple

```
 <include href="fichier.rng"/>
```

Syntaxe : Inclusion simple (compacte)

```
include "fichier.rnc"
```

Syntaxe : Inclusion avec redéfinition de patterns nommés

```
<include href="fichier.rng">
  <define name="ReDefine">
    ...
  </define>
  ...
</include>
<!-- ou ReDefine désigne un élément déjà défini dans  fichier.rng -->
```

Syntaxe : Inclusion avec redéfinition de patterns nommés (compacte)

```
include "fichier.rnc" {
  ...
}
```

Syntaxe : Inclusion avec redéfinition de l'élément racine

```
<include href="fichier.rng">
  <start>
    <ref name="NewStartElement"/>
  </start>
</include>
```

Syntaxe : Inclusion avec redéfinition de l'élément racine (compacte)

```
include "fichier.rnc" {
  start = ...
}
```

7. Types de données

Syntaxe

Ajouter l'attribut `datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"` à la racine `grammar`.

```
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
...
<element name="...">
  <data type="...">
</element>
```

Définition : Types primitifs

- string
- boolean
- decimal
- float, double
- date, dateTime, duration, time, gYearMonth, gYear, gMonthDay, gDay, gMonth
- hexBinary, base64Binary
- anyURI
- QName, NOTATION
- Types hérités des DTD : ID, IDREF, IDREFS...

Complément : Spécification des primitive datatypes

http://www.w3.org/TR/xmlschema-2/

Facette

Paramètre de spécialisation des types primitifs.

Exemple : Type string

Facettes :

- length : longueur de la chaîne
- pattern : expression régulière permettant de valider la chaîne par rapport au patron (définition en intention)
- enumeration : liste de valeurs autorisées (définition en extension)
- ...

Définition : Built-in datatypes

Dérivés des types primitifs.

Par exemple :

- `integer`, dérivé de `decimal`
- `normalizedString`, dérivé de `string`
- `language`, dérivé de `string`
- `ID`, `IDREF`

Exemple : RelaxNG XML

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="mail" datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
      <data type="string">
        <param name="pattern">([^ ])+@([^ ])+.([^ ])+</param>
      </data>
    </element>
  </start>
</grammar>
```

Exemple : RelaxNG compact

```
datatypes xsd="http://www.w3.org/2001/XMLSchema-datatypes"
start = element age {xsd:decimal {maxInclusive="100"}}
```

Complément

http://www.xml.dvint.com/docs/SchemaDataTypesQR-2.pdf

Complément

Guidelines for using W3C XML Schema Datatypes with RELAX NG : *http://relaxng.org/xsd.html*

8. Relax NG : Syntaxe Compacte

Exemple

Le code RelaxNG version compacte suivant permet de spécifier qu'un élément "document" contient des éléments "page" qui contiennent du texte .

```
element document {element page {text}}*
```

Exemple : Syntaxe compacte

Tutoriel *

Syntaxe : Element

```
1 element e1 { element e2 {...
```

Syntaxe : Attributs

```
1 element e2 { attribute a1 {...
```

Syntaxe : Nœuds texte

```
1 element e1 {text}
```

Syntaxe : Cardinalité

```
1 element e1 { element e2 {...}*, element e3 {...}+, ...}
```

Syntaxe : Optionalité

```
1 element e1 { element e2 {...}?,  attribut a1 {...}?}
```

Syntaxe : Alternative

```
1 element e1 { element e2 {...} | element e3 {...}}
```

Exemple : Ensemble non ordonné (set)

```
1 element setElement { (element e2 {...} | element e3 {...})* }
```

Exemple : Contenu mixte (mixed-content)

```
1 element paraTag { (text | element inlineTag {text})* }
```

Syntaxe : Énumération

```
1 attribute a1 { "v1" | "v2" }
```

```
1 element e1 { "v1" | "v2" }
```

Syntaxe : Pattern nommés

```
1 Pattern1 = ...
2 ...
3 element e1 {Pattern1}
```

Syntaxe : Syntaxe générale

```
1 start = element e1 {...}
2 Pattern1 = ...
```

```
3 Pattern2 = ...
```

Syntaxe : Types de données

Spécifier `datatypes xsd="http://www.w3.org/2001/XMLSchema-datatypes"` avant le `start`

```
1 datatypes xsd="http://www.w3.org/2001/XMLSchema-datatypes"
2 start = ...
3 element e1 {xsd:decimal {maxInclusive="100"}}
```

Guidelines for using W3C XML Schema Datatypes with RELAX NG : *http://relaxng.org/xsd.html*

Syntaxe : Namespace

```
1 namespace rng = "http://relaxng.org/ns/structure/1.0"
2 default namespace = "http://foo.bar.org/xml-schemas/mtoSchema"
3 start = ...
```

9. Complément : Principes de conception de schémas

Quelques exemple de principes d'écriture d'un schéma :

- "Poupées russes" : schéma arborescent dans lequel chaque élément est défini localement à son père
  `element e1 {element e2 {...}}`
  NB : Suit la structure des instances, *pas de réutilisation* de *pattern*
- "Tranches de salami" : schéma réseau dans lequel chaque élément est défini globalement
  `element e1 {... E2 ...} / E2={...}`
  NB : *pas de redéfinition* de *pattern*

(Brillant07:89-92) *

10. Pour aller plus loin...

Traduction de la spécification

http://xmlfr.org/oasis/committees/relax-ng/tutorial-20011203-fr.html

Livre en ligne

http://books.xmlschemata.org/relaxng

Eric van der Vlist(2003), Relax NG, O'Reilly.

Exemples

IV

1. Exemple : Carnet d'adresse (XML, DTD, Relax NG)

Exemple : Instance XML

```
<?xml version="1.0" encoding="UTF-8"?><?oxygen RNGSchema="adresse1.rnc" type="compact"?>
<addressBook>
    <card>
        <name>John Smith</name>
        <email>js@example.com</email>
    </card>
    <card>
        <name>Fred Bloggs</name>
        <email>fb@example.net</email>
    </card>
</addressBook>
```

Exemple : DTD

```
<!ELEMENT addressBook (card*)>
<!ELEMENT card (name, email)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Exemple : RelaxNG (version 1 : sans pattern, XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
    <oneOrMore>
        <element name="card">
            <element name="name"><text/></element>
            <element name="email"><text/></element>
        </element>
    </oneOrMore>
</element>
```

Exemple : RelaxNG (version 1 : sans pattern, compacte)

```
element addressBook {
    element card {
        element name { text },
        element email { text }
    }+
}
```

Exemple : RelaxNG (version 2 : avec pattern, XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
    <start>
        <element name="addressBook">
            <zeroOrMore>
                <element name="card"><ref name="cardContent"/></element>
            </zeroOrMore>
        </element>
    </start>
    <define name="cardContent">
        <element name="name"><text/></element>
        <element name="email"><text/></element>
    </define>
</grammar>
```

Exemple : RelaxNG (version 2 : avec pattern, compacte)

```
namespace rng = "http://relaxng.org/ns/structure/1.0"
start = AddressBook
AddressBook = element addressBook { Card* }
Card = element card { Name, Email }
Name = element name { text }
Email = element email { text }
```

Exemple : RelaxNG (version 3 : avec datatype, XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
    <start>
        <element name="addressBook">
            <zeroOrMore>
                <element name="card"><ref name="cardContent"/></element>
            </zeroOrMore>
        </element>
    </start>
    <define name="cardContent">
        <element name="name"><data type="string"/></element>
        <element name="email"><data type="anyURI"/></element>
    </define>
</grammar>
```

Exemple : RelaxNG (version 3 : avec datatype, compacte)

```
namespace rng = "http://relaxng.org/ns/structure/1.0"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
start = AddressBook
AddressBook = element addressBook { Card* }
Card = element card { Name, Email }
Name = element name { xsd:string }
Email = element email { text }
```

Complément

Sources en téléchargement sur le site

[cf. Exemples : XML, DTD, RelaxNG, XSLT]

2. Exemple : Bulletins météo (XML, DTD, RNC, XSD)

Exemple : Instance XML

```
<?xml version="1.0" encoding="UTF-8"?>
<meteo xmlns="http://foo.bar.org/xml-schemas/mtoSchema">
  <obs num="PY54476VZ32">
      <loc>Paris-Montsouris</loc>
      <date>1998-10-22T15:31:05</date>
      <temp unit="celsius">12.3</temp>
      <hygro>88</hygro>
      <nebulo>8</nebulo>
      <anemo>3</anemo>
      <pluvio>6</pluvio>
  </obs>
  <obs num="BM655S55">
      <loc>Pic-du-midi-bigorre</loc>
      <date>1998-10-22T15:33:10</date>
      <temp unit="celsius">3.1</temp>
      <hygro>55</hygro>
      <nebulo>1</nebulo>
      <anemo>48</anemo>
      <pluvio>0</pluvio>
      <message> Anémomètre primaire HS</message>
  </obs>
</meteo>
```

Exemple : DTD (sans typage des données)

```
<!ELEMENT meteo (obs)+>
<!ATTLIST meteo xmlns CDATA #FIXED 'http://foo.bar.org/xml-schemas/mtoSchema'>
<!ELEMENT obs (loc, date, temp?, hygro?, nebulo?, anemo?, pluvio?, message?) >
<!ATTLIST obs num ID #REQUIRED>
<!ELEMENT loc (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT temp (#PCDATA)>
<!ATTLIST temp unit (celsius | farenheight | kelvin) 'celsius' >
<!ELEMENT hygro (#PCDATA)>
<!ELEMENT nebulo (#PCDATA)>
<!ELEMENT anemo (#PCDATA)>
<!ELEMENT pluvio (#PCDATA)>
<!ELEMENT message (#PCDATA)>
```

Exemple : RelaxNG compacte (avec typage grossier des données)

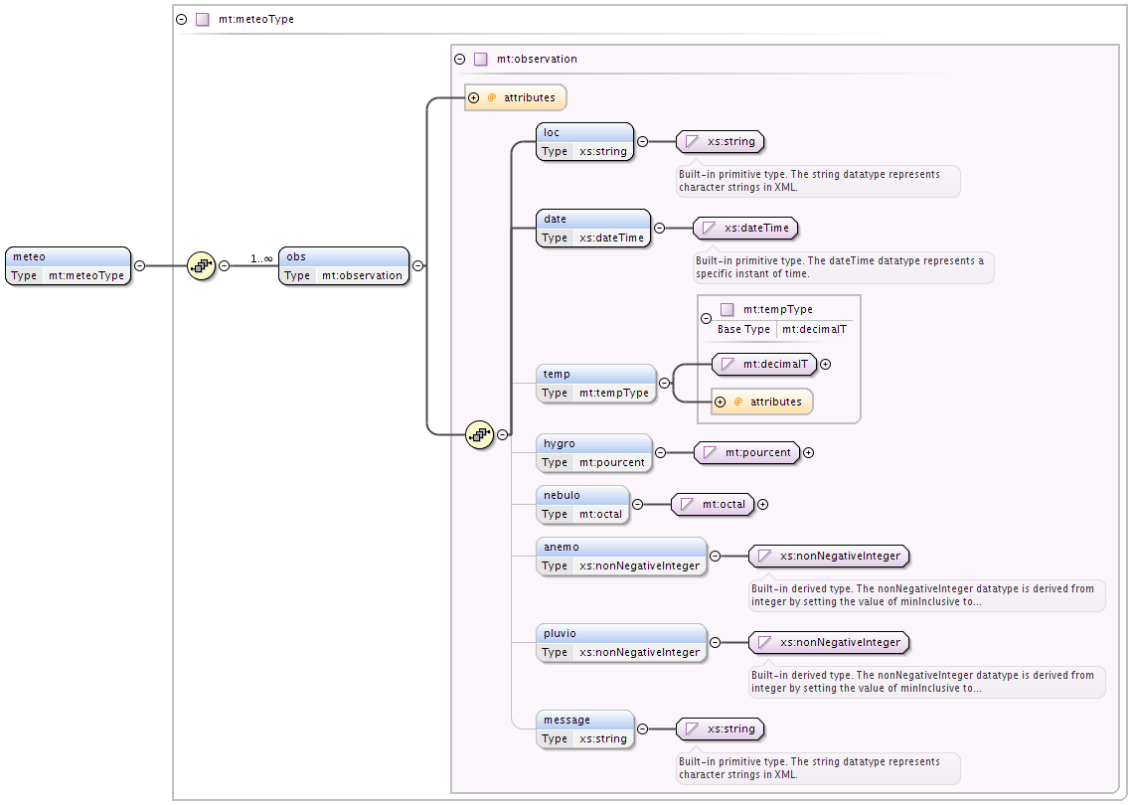
```
namespace rng = "http://relaxng.org/ns/structure/1.0"
default namespace = "http://foo.bar.org/xml-schemas/mtoSchema"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
start= Meteo
```

```
Meteo = element meteo {Obs+}
Obs = element obs {
    element loc {text},
    element date {xsd:dateTime},
    element temp {TempType}?,
    element hygro {Pourcent} ?,
    element nebulo {xsd:decimal}?,
    element anemo {xsd:decimal}?,
    element pluvio {xsd:decimal}?,
    element message {text}?,
    attribute num {xsd:ID}
}
TempType = xsd:decimal, attribute unit {text}
Pourcent = xsd:nonNegativeInteger { maxInclusive = "100" }
```

Exemple : W3C XML Schema (avec typage fin des données)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema
    targetNamespace="http://foo.bar.org/xml-schemas/mtoSchema"
    xmlns="http://foo.bar.org/xml-schemas/mtoSchema"
    xmlns:mt="http://foo.bar.org/xml-schemas/mtoSchema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="meteo" type="mt:meteoType"/>
    <xs:complexType name="meteoType">
        <xs:sequence>
          <xs:element name="obs" type="mt:observation" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="observation">
        <xs:sequence>
          <xs:element name="loc" type="xs:string"/>
          <xs:element name="date" type="xs:dateTime"/>
          <xs:element name="temp" type="mt:tempType" minOccurs="0"/>
          <xs:element name="hygro" type="mt:pourcent" minOccurs="0"/>
          <xs:element name="nebulo" type="mt:octal" minOccurs="0"/>
          <xs:element name="anemo" type="xs:nonNegativeInteger" minOccurs="0"/>
          <xs:element name="pluvio" type="xs:nonNegativeInteger" minOccurs="0"/>
          <xs:element name="message" type="xs:string" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="num" type="xs:ID" use="required"/>
    </xs:complexType>
    <xs:simpleType name="octal">
        <xs:restriction base="xs:nonNegativeInteger">
          <xs:maxInclusive value="8"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="pourcent">
        <xs:restriction base="xs:nonNegativeInteger">
          <xs:maxInclusive value="100"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="decimalT">
        <xs:restriction base="xs:decimal">
          <xs:minInclusive value="-50.0"/>
          <xs:maxInclusive value="+50.0"/>
```

```
41          </xs:restriction>
42      </xs:simpleType>
43      <xs:complexType name="tempType">
44          <xs:simpleContent>
45            <xs:extension base="mt:decimalT">
46              <xs:attribute name="unit" type="mt:tempUnit"/>
47            </xs:extension>
48          </xs:simpleContent>
49      </xs:complexType>
50      <xs:simpleType name="tempUnit">
51        <xs:restriction base="xs:string">
52          <xs:enumeration value="celsius"/>
53          <xs:enumeration value="kelvin"/>
54          <xs:enumeration value="farenheight"/>
55        </xs:restriction>
56      </xs:simpleType>
57  </xs:schema>
```



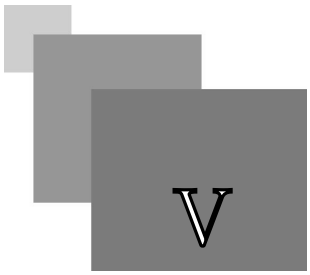
W3C XML Schema visualisé graphiquement dans l'éditeur Oxygen

Complément

Sources en téléchargement sur le site

[cf. Exemples : XML, DTD, W3C Schema, RelaxNG]

Exercices



1. Exercices DTD

Rappel : Manipulation des *Schémas XML* (cf. p.) dans Oxygen

1.1. Exercice : Une DTD

Soit la DTD suivante :

```
1 <!ELEMENT entete (titre, date, auteur+, motscles*, resume?)>
2 <!ELEMENT titre (#PCDATA)>
3 <!ELEMENT date (#PCDATA)>
4 <!ELEMENT auteur (#PCDATA)>
5 <!ELEMENT motscles (#PCDATA)>
6 <!ELEMENT resume (paragraphe+)>
7 <!ELEMENT paragraphe (#PCDATA)>
```

Question 1

[solution n°1 p.46]

Produire un document XML valide par rapport à cette DTD

Question 2

[solution n°2 p.46]

Le document suivant est-il valide par rapport à la DTD ?

```
1 <?xml version="1.0"?>
2 <!DOCTYPE entete SYSTEM "entete.dtd">
3 <entete>
4     <titre>Document de test</titre>
5     <date>2 décembre 2009</date>
6     <auteur>Stéphane Crozat</auteur>
7     <motscles>Document DTD XML Valide</motscles>
8 </entete>
```

Question 3

[solution n°3 p.46]

Produire le plus petit document XML valide par rapport à cette DTD.

Question 4

[solution n°4 p.46]

Produire un document XML utilisant toutes les balises de la DTD.

1.2. Exercice

[solution n°5 p.47*]*

Le fichier `file.xml` n'est pas valide par rapport à la DTD `schema.dtd`. Sélectionnez les éléments causes de cette non-validité.

```
1 <?xml version="1.0"?>
2 <!--file.xml-->
3 <!DOCTYPE papier SYSTEM "schema.dtd">
4 <papier>
5   <titre>Réinterroger les structures documentaires</titre>
6   <auteur>Stéphane Crozat</auteur>
7   <auteur>Bruno Bachimont</auteur>
8   <resume>Nous proposons dans cet article d'aborder ...</resume>
9   <abstract>In this paper we define...</abstract>
10  <motsCles>
11    <terme>Ingénierie des connaissances</terme>
12    <terme>Document</terme>
13  </motsCles>
14  <version num='1'/>
15  <ressource src="sic_00001016.pdf"/>
16 </papier>
```

```
1 <!-- schema.dtd-->
2 <!ELEMENT papier (titre, sousTitre?, auteur, resume, abstract, motsCles, (version | ressource)*)>
3 <!ELEMENT titre (#PCDATA)>
4 <!ELEMENT sousTitre (#PCDATA)>
5 <!ELEMENT auteur (#PCDATA)>
6 <!ELEMENT resume (#PCDATA)>
7 <!ELEMENT motsCles (#PCDATA)>
8 <!ELEMENT version (#PCDATA)>
9 <!ELEMENT ressource EMPTY>
```

sousTitre | resume | titre | auteur | abstract | version | motsCles | ressource

Éléments correctement spécifiés	Éléments incorrectement spécifiés

1.3. Exercice : Utiliser des DTD avec Oxygen

L'objectif de cet exercice est de s'approprier l'environnement Oxygen pour manipuler des DTD.

- Installation Oxygen : *http://stph.crzt.fr/doc/021out/www/co/oxyAC01.html*
- Tutoriel Oxygen pour les schémas : *Schémas XML* (cf. p.)

Question 1

[solution n°6 p.48*]*

Créer une DTD "doc" :

- doc -> p+

- p -> #PCDATA

Question 2

[solution n°7 p.48]

Créer un fichier XML :

- Vérifier qu'il est bien formé
- Faire un test de fichier mal formé

Question 3

[solution n°8 p.48]

Ajouter une référence à la DTD "doc" :

- Valider le fichier
- Faire un test de fichier non valide en modifiant le fichier XML
- Faire un test de fichier non valide en modifiant la DTD

1.4. Exercice :

Pour cet exercice on utilisera un éditeur XML, tel que Oxygen.

Soit la DTD suivante :

```
<!ELEMENT document (entete,corps)>

<!ELEMENT entete (identification, motscles?, resume?)>
<!ELEMENT identification (titre, date, auteur, version)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT motscles (motitem+)>
<!ELEMENT motitem (#PCDATA)>
<!ELEMENT resume (paragraphe+)>

<!ELEMENT corps (introduction?, (div+ | contenu), conclusion?)>
<!ELEMENT div (titre, introduction?, (div+ | contenu), conclusion?)>
<!ELEMENT introduction (paragraphe+)>
<!ELEMENT conclusion (paragraphe+)>
<!ELEMENT contenu (paragraphe)+>

<!ELEMENT paragraphe (#PCDATA | important | etranger | note)*>

<!ELEMENT important (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT etranger (#PCDATA)>
<!ATTLIST etranger
  langue CDATA #IMPLIED
>
```

Question 1

Produire un document XML valide par rapport à cette DTD.

Question 2

Produire le plus petit document XML valide par rapport à cette DTD.

Question 3

Produire un document XML utilisant toutes les balises de la DTD.

Soit le document XML suivant.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE document SYSTEM "document.dtd">
<document>
  <entete>
    <identification>
      <titre>Text</titre>
      <date>Text</date>
      <auteur>Text</auteur>
      <version>Text</version>
    </identification>
    <motscles>
      <motitem>Text</motitem>
      <motitem>Text</motitem>
      <motitem>Text</motitem>
    </motscles>
  </entete>
  <corps>
    <introduction>
      <paragraphe>Text</paragraphe>
      <paragraphe>Text <important>Text</important>Text</paragraphe>
    </introduction>
    <div>
      <titre>Text</titre>
      <introduction>
        <paragraphe>Text</paragraphe>
      </introduction>
      <contenu>
        <paragraphe>Text</paragraphe>
      </contenu>
      <conclusion>
        <paragraphe><note>Text</note></paragraphe>
      </conclusion>
    </div>
    <div>
      <titre>Text</titre>
      <div>
        <titre></titre>
        <contenu>
          <paragraphe>Text</paragraphe>
        </contenu>
      </div>
      <conclusion>
        <paragraphe>Text</paragraphe>
      </conclusion>
    </div>
    <conclusion>
      <paragraphe>Text</paragraphe>
    </conclusion>
```

```
49    </corps>
50 </document>
```

Question 4

Ce document est-il valide par rapport à la DTD ?

2. Quiz DTD

2.1. Préambule

Soit les fichiers XML et DTD suivants :

cours1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cours>
<definition>XML est un méta-langage</definition>
<exemple>XHTML est un langage XML</exemple>
<exemple>SMIL est un langage XML</exemple>
</cours>
```

cours2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cours>
<definition>XML est un méta-langage</definition>
<remarque>XML est hérité de SGML</remarque>
</cours>
```

cours3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cours>
<definition>XML est un méta-langage</definition>
</cours>
<cours>
<exemple>XHTML est un langage XML</exemple>
</cours>
```

cours4.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cours>
<definition>XML est un méta-langage</definition>
<exemple>XHTML est un langage XML</exemple>
<remarque>XML est hérité de SGML</remarque>
</cours>
```

schCours1.dtd

```
<!ELEMENT cours (definition, exemple)>
<!ELEMENT definition (#PCDATA) >
<!ELEMENT exemple (#PCDATA) >
```

schCours2.dtd

```
<!ELEMENT cours (definition, remarque?)>
<!ELEMENT definition (#PCDATA) >
<!ELEMENT remarque (#PCDATA) >
```

2.2. Exercice : Fichiers bien formés

[solution n°9 p.48*]*

Quels sont les fichiers bien formés ?

- ☐ cours1.xml
- ☐ cours2.xml
- ☐ cours3.xml
- ☐ cours4.xml
- ☐ schCours1.dtd
- ☐ schCours2.dtd

2.3. Exercice : Fichiers valides

[solution n°10 p.49*]*

Quels sont les fichiers XML valides par rapport à quelles DTD ?

cours4.xml | cours3.xml | cours1.xml | cours2.xml

Est valide par rapport à `schCours1.dtd`	Est valide par rapport à `schCours2.dtd`	N'est pas valide

2.4. Exercice : Schéma intégrateur

[solution n°11 p.49*]*

Proposer un schéma `schCours3.dtd` tel que tous les fichiers bien formés parmi `cours1.xml`, `cours2.xml`, `cours3.xml` et `cours4.xml` soient valides.

<!ELEMENT cours (, , ,)>

<!ELEMENT definition (#PCDATA) >

<!ELEMENT exemple (#PCDATA) >

<!ELEMENT remarque (#PCDATA) >

3. Exercices RelaxNG

Rappel : Manipulation des *Schémas XML* (cf. p.) dans Oxygen

3.1. Exercice :

Soit le schéma S1 ci-après.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <grammar
3  xmlns="http://relaxng.org/ns/structure/1.0"
4  ns="http://utc.fr/nf29/S1">
5 <start>
6 <element name="book">
7   <optional>
8   <element name="info">
9     <element name="title"><text/></element>
10    <optional>
11    <element name="author">
12      <element name="email"><text/></element>
13    </element>
14    </optional>
15   </element>
16  </optional>
17  <oneOrMore>
18  <element name="part">
19    <ref name="Title"/>
20    <choice>
21      <ref name="Text"></ref>
22      <oneOrMore>
23       <element name="chapter">
24         <ref name="Title"/>
25         <ref name="Text"></ref>
26       </element>
27      </oneOrMore>
28    </choice>
29  </element>
30  </oneOrMore>
31 </element>
32 </start>
33 <define name="Title">
34   <element name="title"><text/></element>
35   <optional>
36   <element name="subtitle"><text/></element>
37   </optional>
38 </define>
39 <define name="Text">
40    <oneOrMore>
41    <element name="para"><text/></element>
42    </oneOrMore>
43 </define>
44 </grammar>
```

Question 1

[solution n°12 p.49]

Produisez le plus petit document XML valide possible.

Question 2

[solution n°13 p.50]

Produisez le plus petit document XML valide contenant tous les éléments.

3.2. Exercice :

Soit les 3 fichiers RelaxNG suivants.

```
<!-- db0.rng -->
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<define name="Title">
  <element name="title"><text/></element>
</define>
<define name="Text">
  <oneOrMore>
    <element name="para"><text/></element>
  </oneOrMore>
</define>
</grammar>
```

```
<!-- db1.rng -->
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
 <element name="book">
  <oneOrMore>
    <element name="part">
      <ref name="Title"/>
      <ref name="Text"></ref>
    </element>
  </oneOrMore>
 </element>
</start>
<include href="db0.rng"></include>
</grammar>
```

```
<!-- db2.rng -->
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
 <element name="book">
  <optional>
    <element name="info">
      <element name="title"><text/></element>
    </element>
  </optional>
  <zeroOrMore>
    <element name="part">
      <ref name="Title"/>
      <ref name="Text"></ref>
      <zeroOrMore>
        <element name="chapter">
          <ref name="Title"/>
          <ref name="Text"></ref>
        </element>
      </zeroOrMore>
    </element>
  </zeroOrMore>
</element>
</start>
```

```
<include href="db0.rng"></include>
</grammar>
```

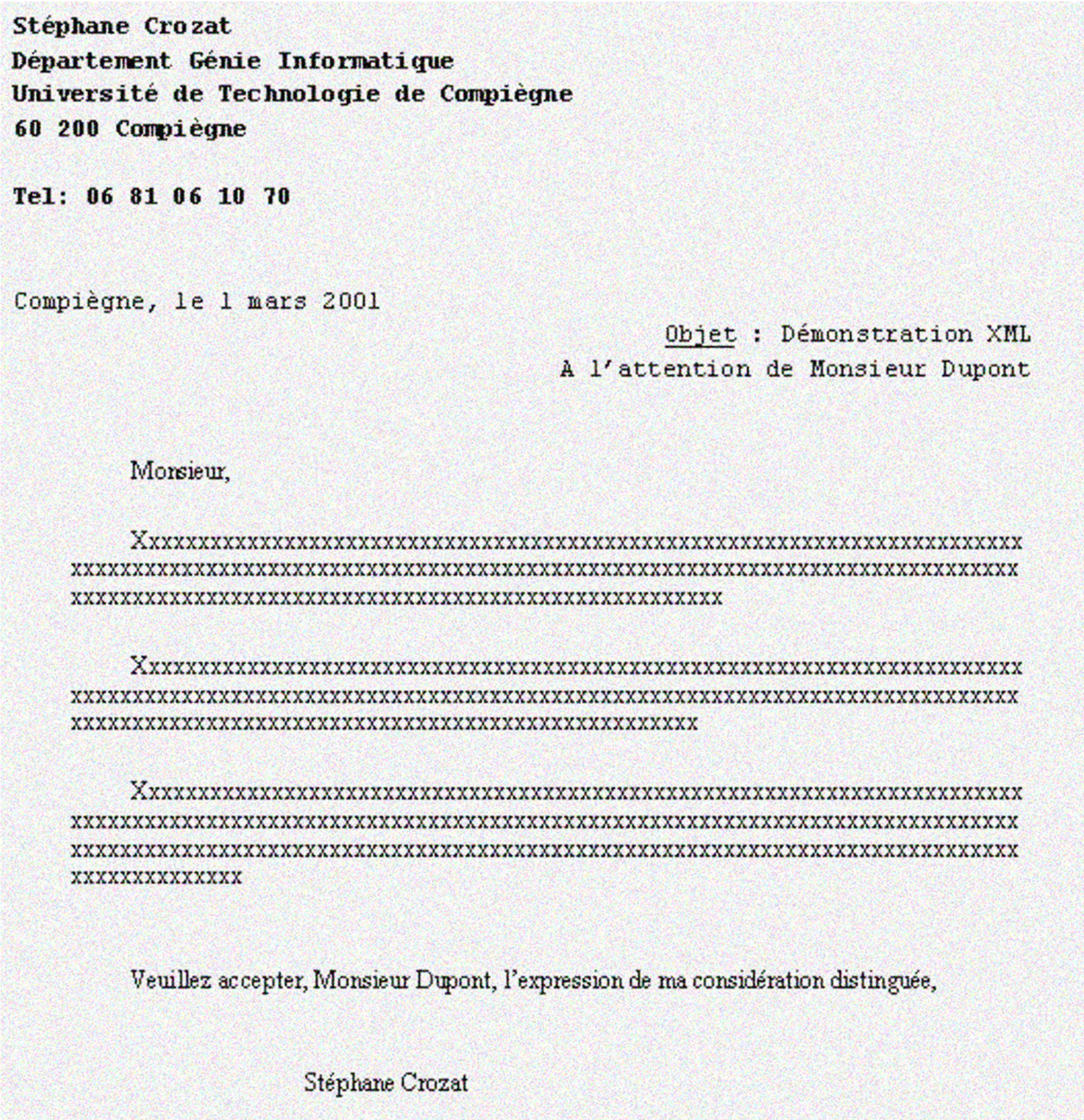
Question

[solution n°14 p.50]

Proposer un fichier `db.rng` permettant de valider à la fois les fichiers valides par rapport à `db1.rng` et `db2.rng`.

3.3. Exercice :

Représenter la lettre officielle suivante en utilisant le formalisme XML.



Stéphane Crozat
Département Génie Informatique
Université de Technologie de Compiègne
60 200 Compiègne

Tel: 06 81 06 10 70

Compiègne, le 1 mars 2001

Objet : Démonstration XML
A l'attention de Monsieur Dupont

Monsieur,

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxx

Veuillez accepter, Monsieur Dupont, l'expression de ma considération distinguée,

Stéphane Crozat

Une lettre

Question 1

[solution n°15 p.50]

Repérer la structure physique du document.

Question 2

[solution n°16 p.51]

Déduire une structure logique depuis la structure physique et écrire le document XML correspondant.

Question 3

[solution n°17 p.52*]*

Écrire le schéma RelaxNG des lettres respectant ce modèle.

Indice :

Utiliser :

- Des *patterns*, par exemple `Texte = paragraphe+`
- Des *datatype*, par exemple pour la date
- Proposer votre propre *namespace*

3.4. Exercice

[solution n°18 p.53*]*

Compléter le schéma RelaxNG `schema.rng` afin que les fichiers XML `file1.xml`, `file2.xml` et `file3.xml` soient valides. On cherchera le schéma *le plus restrictif* possible. On cherchera un schéma sans description redondante, en utilisant la syntaxe `ref/define` lorsque c'est nécessaire.

```
1 <?xml version="1.0"?>
2 <!--file1.xml-->
3 <?oxygen RNGSchema="04.rng" type="xml"?>
4 <a>
5 <b><b1/></b>
6 </a>
```

```
1 <?xml version="1.0"?>
2 <!--file2.xml-->
3 <?oxygen RNGSchema="04.rng" type="xml"?>
4 <a>
5 <b><b1/></b>
6 <b><b1/><b2/></b>
7 <b><b1/></b>
8 </a>
```

```
1 <?xml version="1.0"?>
2 <!--file3.xml-->
3 <?oxygen RNGSchema="04.rng" type="xml"?>
4 <a>
5 <b><b1/></b>
6 <b><b1/><b2/></b>
7 <b><b1/></b>
8 <c/>
9 <c/>
10 <b><b1/></b>
11 <b><b1/></b>
12 </a>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<                xmlns="http://relaxng.org/ns/structure/1.0">
<                >
  <element name="              ">
    <              >
      <ref name="B"/>
    </              >
    <              >
      <element name="              "><              /></element>
    </              >
    <              >
      <ref name="              "/>
    </              >
  </element>
```

```
</                >
<                 name="                ">
  <element name="                ">
    <element name="                "><                /></element>
    <                >
      <element name="                "><                /></element>
    </                >
  </element>
</                >
</grammar>
```

3.5. Exercice : Schéma CV

Vous travaillez dans une agence d'intérim. Votre mission est de mettre en place une chaîne éditoriale XML, afin de gérer les CV de votre agence.

Vous avez à votre disposition pour cela un exemple de CV.

[cf. CV exemple]

Question 1

[solution n°19 p.55*]*

Ré-écrivez ce CV en XML.

Question 2

Proposez un schéma XML.

Question 3

Expliquez en quoi l'approche suivie ici est en général insuffisante pour aboutir à un bon schéma (en notant que l'on ne dispose que d'un seul CV à analyser). Proposez des exemples d'actions qui auraient permis de faire mieux.

Solutions des exercices

> **Solution** n°1

Exercice p. 33

Exemple

```
1 <?xml version="1.0"?>
2 <!DOCTYPE entete SYSTEM "entete.dtd">
3 <entete>
4       <titre>Mon document</titre>
5       <date>Aujourd'hui</date>
6       <auteur>Moi</auteur>
7 </entete>
```

> **Solution** n°2

Exercice p. 33

Oui.

> **Solution** n°3

Exercice p. 33

```
1 <?xml version="1.0"?>
2 <!DOCTYPE entete SYSTEM "entete.dtd">
3 <entete>
4       <titre/>
5       <date/>
6       <auteur/>
7 </entete>
```

> **Solution** n°4

Exercice p. 33

Exemple

```
1 <?xml version="1.0"?>
2 <!DOCTYPE entete SYSTEM "entete.dtd">
3 <entete>
4       <titre>Mon document</titre>
5       <date>Aujourd'hui</date>
6       <auteur>Moi</auteur>
7       <motscles>Document</motscles>
```

```
    <resume>
        <paragraphe>Mon résumé</paragraphe>
    </resume>
</entete>
```

> **Solution** n°5

Exercice p. 33

Le fichier `file.xml` n'est pas valide par rapport à la DTD `schema.dtd`. Sélectionnez les éléments causes de cette non-validité.

```
<?xml version="1.0"?>
<!--file.xml-->
<!DOCTYPE papier SYSTEM "schema.dtd">
<papier>
  <titre>Réinterroger les structures documentaires</titre>
  <auteur>Stéphane Crozat</auteur>
  <auteur>Bruno Bachimont</auteur>
  <resume>Nous proposons dans cet article d'aborder ...</resume>
  <abstract>In this paper we define...</abstract>
  <motsCles>
    <terme>Ingénierie des connaissances</terme>
    <terme>Document</terme>
  </motsCles>
  <version num='1'/>
  <ressource src="sic_00001016.pdf"/>
</papier>
```

```
<!-- schema.dtd-->
<!ELEMENT papier (titre, sousTitre?, auteur, resume, abstract, motsCles, (version | ressource)*)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT sousTitre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT resume (#PCDATA)>
<!ELEMENT motsCles (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT ressource EMPTY>
```

Éléments correctement spécifiés	Éléments incorrectement spécifiés
titre	auteur
sousTitre	abstract
resume	motsCles
	version
	ressource

Les erreurs :

- Il manque la cardinalité N sur auteur (`auteur*` ou `auteur+`)

- Il manque la déclaration de `abstract` dans `papier` et sa définition
- `motsCles` n'est pas du texte, mais contient des éléments `terme`, qui doivent être définis.
- Il manque la définition des attributs de `version` et `ressource`.

Ci-après la DTD corrigée.

```
<!-- schema.dtd-->
<!ELEMENT papier (titre, sousTitre?, auteur*, resume, abstract, motsCles, (version | ressource)*)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT sousTitre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT resume (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT motsCles (terme+)>
<!ELEMENT terme (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ATTLIST version num CDATA #IMPLIED>
<!ELEMENT ressource EMPTY>
<!ATTLIST ressource src CDATA #REQUIRED>
```

> **Solution** n°6

Exercice p. 34

```
<!ELEMENT doc (p+)>
<!ELEMENT p (#PCDATA) >
```

> **Solution** n°7

Exercice p. 35

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
<p>Hello world !</p>
</doc>
```

> **Solution** n°8

Exercice p. 35

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE doc SYSTEM "doc.dtd">
<doc>
<p>Hello world !</p>
</doc>
```

> **Solution** n°9

Exercice p. 39

Quels sont les fichiers bien formés ?

- ☑ cours1.xml
- ☑ cours2.xml

- ☐ cours3.xml
- ☑ cours4.xml
- ☐ schCours1.dtd
- ☐ schCours2.dtd

Les DTD ne sont pas des fichiers XML (il n'y a pas de balise), ils ne sont donc pas bien formés.

Un fichier est bien formé s'il respecte les règles syntaxiques générales de XML *et* s'il possède un *unique* élément racine *et* si tous les éléments qu'il contient sont totalement imbriqués les uns dans les autres (tout élément n'a qu'un seul père, les balises de se "croisent" pas).

> **Solution** n°10

Exercice p. 39

Quels sont les fichiers XML valides par rapport à quelles DTD ?

Est valide par rapport à `schCours1.dtd`	**Est valide par rapport à `schCours2.dtd`**	**N'est pas valide**
	cours2.xml	cours1.xml
		cours3.xml
		cours4.xml

> **Solution** n°11

Exercice p. 39

Proposer un schéma `schCours3.dtd` tel que tous les fichiers bien formés parmi `cours1.xml`, `cours2.xml`, `cours3.xml` et `cours4.xml` soient valides.

<!ELEMENT cours (`definition`, `exemple *`, `remarque ?`)>

<!ELEMENT definition (#PCDATA) >

<!ELEMENT exemple (#PCDATA) >

<!ELEMENT remarque (#PCDATA) >

- `cours3.xml` n'est pas concerné car il n'est pas bien formé, un fichier ne peut être valide s'il n'est pas bien formé.
- On peut accepter `definition?` ou `definition*`, ainsi que `remarque*`, même si ces cardinalités ne sont pas instanciées pas dans les fichiers XML cités en exemple.

> **Solution** n°12

Exercice p. 40

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
<book xmlns="http://utc.fr/nf29/S1">
  <part>
    <title/>
    <para/>
  </part>
</book>
```

> **Solution** n°13

Exercice p. 40

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://utc.fr/nf29/S1">
  <info>
    <title/>
    <author>
      <email/>
    </author>
  </info>
  <part>
    <title/>
    <subtitle/>
    <chapter>
      <title/>
      <para/>
    </chapter>
  </part>
</book>
```

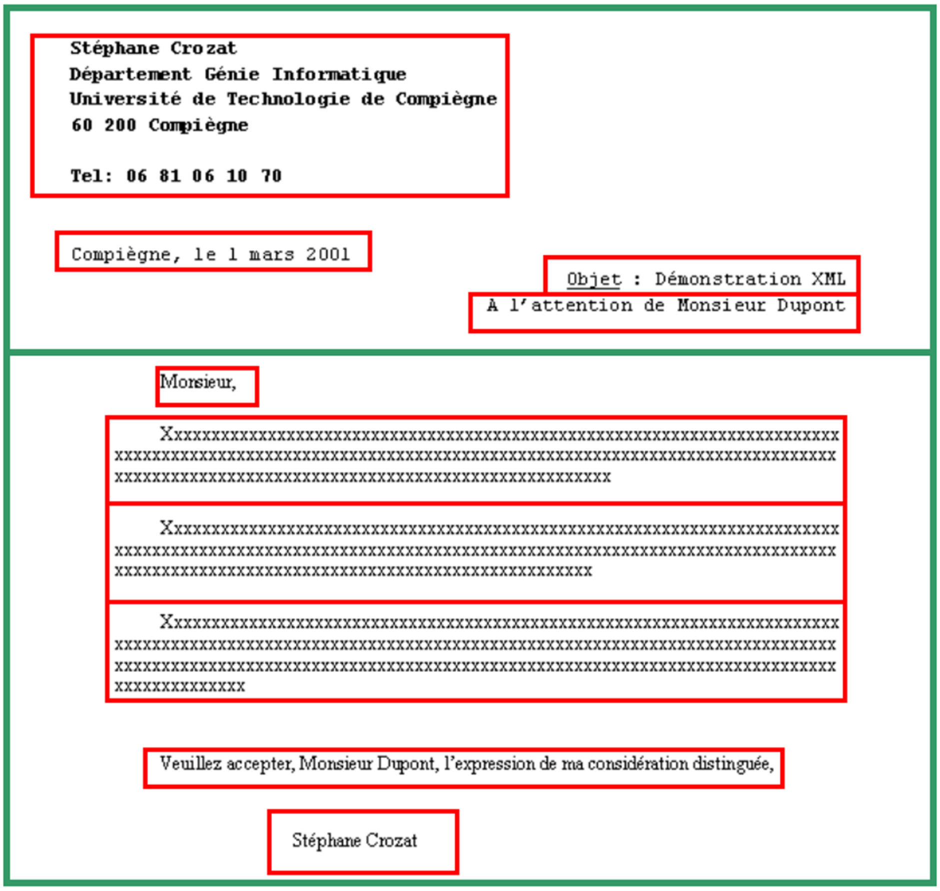
> **Solution** n°14

Exercice p. 42

`db2.rng` permet de valider les fichiers `db1.rng`, donc `db.rng=db2.rng`.

> **Solution** n°15

Exercice p. 42


Stéphane Crozat
Département Génie Informatique
Université de Technologie de Compiègne
60 200 Compiègne

Tel: 06 81 06 10 70

Compiègne, le 1 mars 2001

Objet : Démonstration XML
A l'attention de Monsieur Dupont

Monsieur,

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Xxx

Veuillez accepter, Monsieur Dupont, l'expression de ma considération distinguée,

Stéphane Crozat

Structure physique de la lettre XML

> **Solution** n°16

Exercice p. 42

Exemple

```
<?xml version="1.0"?>
<lettre>
    <entete>
        <emetteur>
            <prenom>Stéphane</prenom>
            <nom>Crozat</nom>
            <adresse>
                <texte>
                    <paragraphe>UTC</paragraphe>
                    <paragraphe>60200 Compiègne</paragraphe>
                </texte>
            </adresse>
        </emetteur>
        <destinataire>
            <titre>Monsieur</titre>
```

```
            <nom>Dupont</nom>
        </destinataire>
        <date>2001-03-01</date>
        <lieu>Compiègne</lieu>
        <objet>Démonstration XML</objet>
    </entete>
    <corps>
        <texte>
            <paragraphe>XXXXXXXXXXX...</paragraphe>
            <paragraphe>XXXXXXXXXXX...</paragraphe>
            <paragraphe>XXXXXXXXXXX...</paragraphe>
        </texte>
    s</corps>
</lettre>
```

Remarque : Quelques remarques

La structure logique est très éloignée de la mise en forme dans la page. En particulier aucune information n'est présente sur la localisation dans la page des informations, c'est la structure d'une lettre qui les prescrira à partir des descripteurs logiques.

On note que certains éléments, s'ils sont calculables, n'ont pas besoins d'être spécifiés dans la structure logique, comme ici la formule de politesse et la signature.

> **Solution** n°17

Exercice p. 43

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
    ns="www.utc.fr/nf29/tdXml">
    <start>
        <element name="lettre">
            <element name="entete">
                <element name="emetteur">
                    <ref name="personne"/>
                </element>
                <element name="destinataire">
                    <ref name="personne"></ref>
                </element>
                <element name="date"><data type="date"></data></element>
                <element name="lieu"><text/></element>
                <element name="objet"><text/></element>
            </element>
            <element name="corps">
                <ref name="texte"/>
            </element>
        </element>
    </start>
    <define name="personne">
        <optional>
            <element name="titre"><text/></element>
        </optional>
        <zeroOrMore>
            <element name="prenom" ><text/></element>
```

```
30          </zeroOrMore>
31              <element name="nom"><text/></element>
32          <optional>
33          <element name="adresse">
34              <ref name="texte"/>
35          </element>
36          </optional>
37      </define>
38      <define name="texte">
39          <element name="texte">
40              <oneOrMore>
41                  <element name="paragraphe"><text/></element>
42              </oneOrMore>
43          </element>
44      </define>
45 </grammar>
```

[cf. Schéma RelaxNG d'une lettre]

> **Solution** n°18

Exercice p. 44

Compléter le schéma RelaxNG `schema.rng` afin que les fichiers XML `file1.xml`, `file2.xml` et `file3.xml` soient valides. On cherchera le schéma *le plus restrictif* possible. On cherchera un schéma sans description redondante, en utilisant la syntaxe `ref/define` lorsque c'est nécessaire.

```
1 <?xml version="1.0"?>
2 <!--file1.xml-->
3 <?oxygen RNGSchema="04.rng" type="xml"?>
4 <a>
5 <b><b1/></b>
6 </a>
```

```
1 <?xml version="1.0"?>
2 <!--file2.xml-->
3 <?oxygen RNGSchema="04.rng" type="xml"?>
4 <a>
5 <b><b1/></b>
6 <b><b1/><b2/></b>
7 <b><b1/></b>
8 </a>
```

```
1 <?xml version="1.0"?>
2 <!--file3.xml-->
3 <?oxygen RNGSchema="04.rng" type="xml"?>
4 <a>
5 <b><b1/></b>
6 <b><b1/><b2/></b>
7 <b><b1/></b>
8 <c/>
9 <c/>
10 <b><b1/></b>
11 <b><b1/></b>
12 </a>
```

<?xml version="1.0" encoding="UTF-8"?>

<`grammar` xmlns="http://relaxng.org/ns/structure/1.0">

```
<start>
  <element name="a">
    <oneOrMore>
      <ref name="B"/>
    </oneOrMore>
    <zeroOrMore>
      <element name="c"><empty/></element>
    </zeroOrMore>
    <zeroOrMore>
      <ref name="B"/>
    </zeroOrMore>
  </element>
</start>
<define name="B">
  <element name="b">
    <element name="b1"><empty/></element>
    <optional>
      <element name="b2"><empty/></element>
    </optional>
  </element>
</define>
</grammar>
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <grammar xmlns="http://relaxng.org/ns/structure/1.0">
3  <start>
4    <element name="a">
5      <oneOrMore>
6        <ref name="B"/>
7      </oneOrMore>
8      <zeroOrMore>
9        <element name="c"><empty/></element>
10     </zeroOrMore>
11     <zeroOrMore>
12       <ref name="B"/>
13     </zeroOrMore>
14   </element>
15 </start>
16 <define name="B">
17   <element name="b">
18     <element name="b1"><empty/></element>
19     <optional>
20       <element name="b2"><empty/></element>
```

```
    </optional>
  </element>
</define>
</grammar>
```

> **Solution** n°19

Exercice p. 45

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<cv xmlns="utc.fr/nf29/crozatst/schema/cv">
  <entete>
    <nom>Scotto</nom>
    <prenom>Léo</prenom>
    <ddn>18/07/1985</ddn>
    <nationalite>Française</nationalite>
    <situationMaritale>Célibataire</situationMaritale>
    <permis>B</permis>
    <mail>leo@monmail.fr</mail>
    <web>http://leo.monsiteperso.fr</web>
  </entete>
  <situationActuelle>Étudiant en première année de philosophie à l'université Upsilon</situationActuelle>
  <diplomes>
    <diplome>
      <annee>2006</annee>
      <intitule>Baccalauréat littéraire</intitule>
    </diplome>
    <diplome>
      <annee>2003</annee>
      <intitule>Brevet des collèges</intitule>
    </diplome>
  </diplomes>
  <stages>
    <stage>
      <date>été</date>
      <annee>2005</annee>
      <titre>Relecteur chez Eyrolles</titre>
      <description>Révision d'ouvrages d'informatique et utilisation de la suite OpenOffice.org et du
logiciel de
        gestion de versions Subversion.</description>
    </stage>
    <stage>
      <date>été</date>
      <annee>2004</annee>
      <titre>Stagiaire à l'UTC</titre>
      <description>Structuration documentaire, utilisation de Scenari.</description>
    </stage>
  </stages>
  <competences>
    <competence>
      <titre>Très bonne pratique du français écrit</titre>
    </competence>
    <competence>
      <titre>Outils bureautiques</titre>
      <description>Suite OpenOffice.org, Adobe Photoshop et The Gimp, Adobe
        Illustrator et Inkscape</description>
    </competence>
```

```
48      </competences>
49      <langues>
50          <langue>
51              <intitule>Anglais</intitule>
52              <niveau>Bilingue</niveau>
53          </langue>
54          <langue>
55              <intitule>Chinois</intitule>
56              <niveau>Rudiments</niveau>
57          </langue>
58      </langues>
59      <loisirs>
60          <loisir>Surf</loisir>
61          <loisir>Cinéma</loisir>
62          <loisir>Roller</loisir>
63          <loisir>Lecture</loisir>
64      </loisirs>
65  </cv>
66
67
68
```

Abréviations

DTD : Document Type Definition

XML : eXtensible Markup Language

Bibliographie

Alexandre Brillant, *XML : Cours et exercices*, Eyrolles, 2007 [ISBN 978-2212126914]

Webographie

James Clark, John Cowan, Murata Makoto, *RELAX NG Compact Syntax Tutorial*, OASIS, 2003. [http://www.relaxng.org/compact-tutorial-20030326.html]

Index