Les schémas pour XML

Fabrice Rossi

http://apiacoa.org/contact.html.

Université Paris-IX Dauphine

Limitations des DTD

Les DTD posent de nombreux problèmes :

- syntaxe non XML
- pas de support des *namespaces*
- modularité assez limitée
- modèle limité :
  - contraintes de structures assez simplistes (nombre, modèle mixte, etc.)
  - contraintes presque inexistantes pour les données (texte et attributs)

Les schémas

Il existe de nombreuses propositions pour remplacer les DTD.
Deux grandes technologies :

- les schémas du W3C :
  - \+ norme officielle
  - \+ modèle de contenu très évolué et très riche
  - \- verbeux
  - \- très complexe
- RELAX NG (consortium OASIS) :
  - \+ assez compact et simple
  - \+ relativement modulaire
  - \- modèle de contenu basique (en version de base)
  - \- moins officiel
- points communs :
  - obligatoirement défini dans un document externe
  - utilisent les *namespaces*
  - syntaxe XML classique (pas de structure à la DTD)

RELAX NG

RELAX NG

- spécification du 12 Décembre 2001
- `http://www.oasis-open.org/committees/relax-ng/`
- implantation Open Source disponible (en Java) : Jing (`http://www.thaiopensource.com/relaxng/jing.html`)
- buts :
  - syntaxe XML
  - support des *namespaces*
  - meilleur modèle de structure qu'avec les DTD (prise en compte du contexte, modèle non ordonné)
  - support pour des types de données complexes (non inclus par défaut)

Exemple : liste de personnes

On veut construire un schéma pour le fichier :

liste-personnes.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <liste>
3  <personne>
4  <prénom>John</prénom><nom>Doe</nom>
5  </personne>
6  <personne>
7  <prénom>Robert</prénom><nom>Durand</nom>
8  </personne>
9  </liste>
```

On souhaite avoir :

- au moins une personne
- exactement un nom
- exactement un prénom

Arbre correspondant

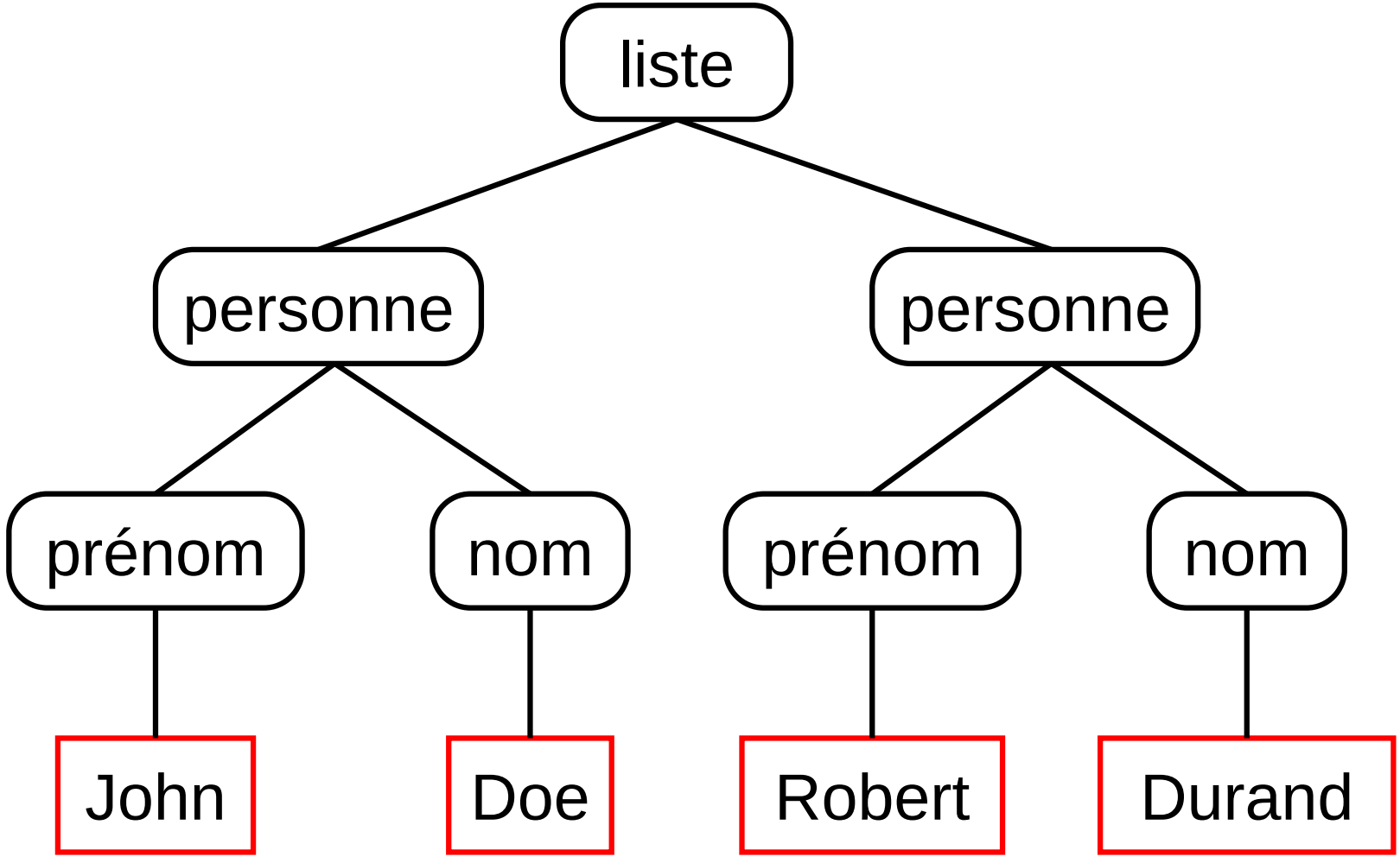
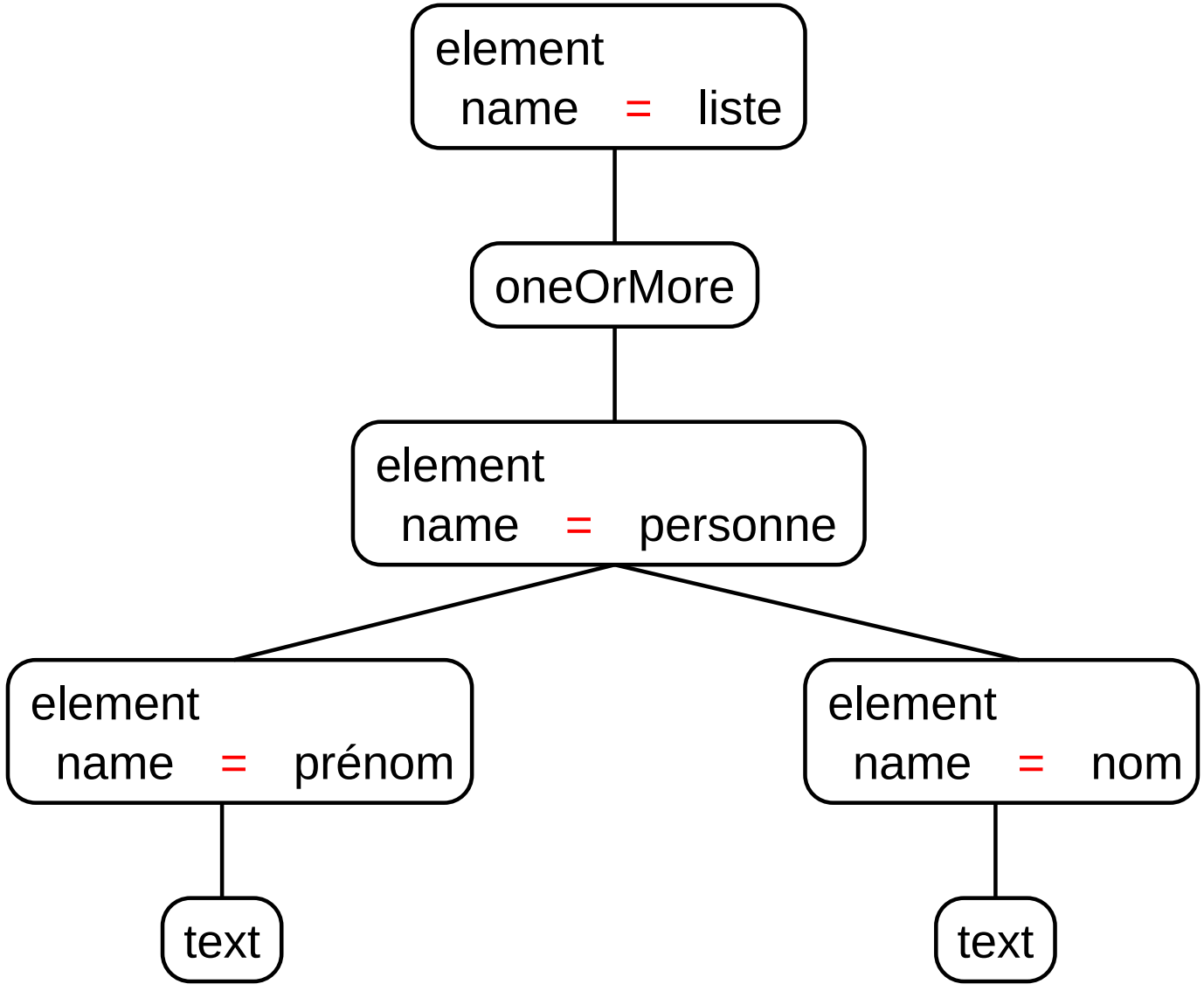


Schéma correspondant

personnes-1.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="liste"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <oneOrMore>
5      <element name="personne">
6        <element name="prénom">
7          <text/>
8        </element>
9        <element name="nom">
10         <text/>
11       </element>
12     </element>
13   </oneOrMore>
14 </element>
```

Arbre du schéma



Remarques

- tous les éléments RELAX NG doivent être dans l'espace de nom `http://relaxng.org/ns/structure/1.0`
- la structure d'un schéma simple reproduit celle d'un exemple de document valide
- les éléments servent à définir des *patterns* (des modèles) qui seront comparés au document pour savoir si celui-ci est valide
- éléments intéressants :
  - `element` : déclaration d'un élément (attribut `name` : nom de l'élément)
  - gestion des répétitions :
    - par défaut : un exactement
    - `oneOrMore` : un au moins
    - `zeroOrMore` : nombre arbitraire
    - `optionnal` : un au plus
- `text` : du texte

DTD équivalente

personnes-1.dtd

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!ELEMENT liste (personne*)>
3  <!ELEMENT personne (prénom, nom)>
4  <!ELEMENT prénom (#PCDATA)>
5  <!ELEMENT nom (#PCDATA)>
```

- plus compacte
- demande une modification du fichier XML (association fichier vers DTD)

Modification du modèle

On souhaite permettre l'utilisation d'un ou deux prénoms.
Solution barbare :

personnes-2.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="liste"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <oneOrMore>
5      <element name="personne">
6        <element name="prénom"><text/></element>
7        <optional>
8          <element name="prénom"><text/></element>
9        </optional>
10       <element name="nom"><text/></element>
11     </element>
12   </oneOrMore>
13 </element>
```

Rappel DTD :

```
<!ELEMENT personne (prénom, prénom?, nom)>
```

Déclarations nommées

Principes :

- au lieu de commencer par un `element`, on commence par un `grammar`
- contient un unique élément `start` qui contient le modèle de la racine du document
- des éléments `define` qui définissent des *patterns*
- chaque `define` à un nom (attribut `name`)
- on peut demander l'inclusion du contenu d'un `define` n'importe où grâce à un élément `ref` (dont le `name` précise le `define` à inclure)

Exemple

personnes-nom.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <grammar xmlns="http://relaxng.org/ns/structure/1.0">
3    <start>
4      <element name="liste">
5        <oneOrMore>
6          <element name="personne">
7            <ref name="Prénom"/>
8            <optional><ref name="Prénom"/></optional>
9            <element name="nom"><text/></element>
10         </element>
11       </oneOrMore>
12     </element>
13   </start>
14   <define name="Prénom">
15     <element name="prénom"><text/></element>
16   </define>
17 </grammar>
```

Autres constructions

- `group` : crée un *pattern* en regroupant d'autres *patterns* (ses fils)
- `choice` : crée une alternative entre des *patterns*
- `interleave` : crée un *pattern* d'entrelacement, i.e., permettant un mélange entre les *patterns* fils :
  - version simple : suppression de l'ordre (autoriser a suivi de b comme b suivi de a)
  - version évoluée : autoriser un nombre précis d'occurrences de certains *patterns*, mais dans n'importe quel ordre (a a b, a b a ou b a a)
- `empty` : pour un élément vide

Suppression de la contrainte d'ordre

Autoriser nom et prénom dans n'importe quel ordre :

personnes-no-order.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="liste"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <oneOrMore>
5      <element name="personne">
6        <interleave>
7          <element name="prénom"><text/></element>
8          <element name="nom"><text/></element>
9        </interleave>
10     </element>
11   </oneOrMore>
12 </element>
```

Suppression de la contrainte d'ordre

Attention aux subtilités :

personnes-no-order-nom.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <grammar xmlns="http://relaxng.org/ns/structure/1.0">
3    <start>
4      <element name="liste">
5        <oneOrMore>
6          <element name="personne">
7            <interleave>
8              <group><ref name="Prénom"/>
9                <optional><ref name="Prénom"/></optional>
10             </group>
11             <element name="nom"><text/></element>
12           </interleave>
13         </element>
14       </oneOrMore>
15     </element>
16   </start>
17   <define name="Prénom"><element name="prénom"><text/></element></define>
18 </grammar>
```

Exemple

Document valide (deuxième schéma) :

liste-personnes-no-order.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <liste>
3  <personne><nom>Doe</nom><prénom>John</prénom></personne>
4  <personne>
5  <prénom>Maurice</prénom><nom>Durand</nom><prénom>Robert</prénom>
6  </personne>
7  <personne>
8  <prénom>Ursula</prénom><prénom>Marie</prénom><nom>Dupont</nom>
9  </personne>
10 <personne>
11 <nom>Dupont</nom><prénom>Ursula</prénom><prénom>Marie</prénom>
12 </personne>
13 <personne>
14 <nom>Martin</nom><prénom>Fernande</prénom><prénom>Germaine</prénom>
15 </personne>
16 <personne><prénom>Alfred</prénom><nom>Blanc</nom></personne>
17 </liste>
```

Modèle mixte évolué

Le modèle mixte est beaucoup plus riche qu'avec les DTDs.
Par exemple des adresses de la forme suivante :

adresses.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <adresses>
3    <adresse>
4      <num>15</num> <rue/> de belleville <code>75020</code>
5      <ville>Paris</ville>
6    </adresse>
7  </adresses>
```

- `num` est facultatif
- `rue` est une possibilité parmi d'autres (`avenue`, etc.)

Impossible à faire avec une DTD.

Un schéma possible

adresses.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="adresses"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <oneOrMore>
5      <element name="adresse">
6        <optional><element name="num"><text/></element></optional>
7        <choice>
8          <element name="rue"><empty/></element>
9          <element name="avenue"><empty/></element>
10         <element name="boulevard"><empty/></element>
11         <element name="place"><empty/></element>
12       </choice>
13       <text/>
14       <element name="code"><text/></element>
15       <element name="ville"><text/></element>
16     </element>
17   </oneOrMore>
18 </element>
```

Attributs

Nouvelle liste de personnes :

liste-personnes-tel.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <liste>
3  <personne>
4  <nom>John Doe</nom>
5  <tel type="fixe">07 78 32 48 54</tel>
6  </personne>
7  <personne>
8  <nom>Robert Durand</nom>
9  <tel type="portable">09 78 32 48 54</tel>
10 <tel type="fixe">07 54 02 94 13</tel>
11 </personne>
12 </liste>
```

Contraintes :

- nombre arbitraire de numéros
- attribut type : nature du numéro

Attributs

- l'élement `attribute` crée un *pattern* correspondant à un attribut (nom de l'attribut donné dans l'attribut `name`)
- `optional` : attribut optionnel
- `choice` : alternative entre attributs
- `group` : groupe d'attributs
- On peut mélanger des *patterns* éléments et attributs dans un groupe et donc dans une alternative.
- `attribute` vide : correspond à un contenu texte pour l'attribut

Attributs

personnes-tel.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="liste"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <oneOrMore>
5      <element name="personne">
6        <element name="nom"><text/></element>
7        <zeroOrMore>
8          <element name="tel">
9            <attribute name="type">
10             <text/>
11           </attribute>
12           <text/>
13         </element>
14       </zeroOrMore>
15     </element>
16   </oneOrMore>
17 </element>
```

Modèles de données

- partie très limitée de RELAX NG
- en version de base, trois types :
  - `text`
  - énumérations (grâce à `choice` et un élément `value`)
  - listes (d'énumérations)
- un mécanisme permet d'utiliser des types définis en dehors de la spécification de RELAX NG (par exemple ceux des schémas du W3C) :
  - l'élément `data` précise le type la donnée grâce à son attribut `type` (et des éléments `param`)
  - l'attribut `datatypeLibrary` d'un *pattern* précise quelle bibliothèque de type de données il utilise
  - certains validateurs (Jing par exemple) permettent de programmer simplement des validateurs pour des types définis par l'utilisateur (cf exercices, `http://apiacoa.org/teaching/xml/exercices/dtd-et-schemas.pdf`)

Enumération

personnes-tel2.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="liste"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <oneOrMore>
5      <element name="personne"><element name="nom"><text/></element>
6        <zeroOrMore>
7          <element name="tel">
8            <attribute name="type">
9              <choice>
10               <value>portable</value>
11               <value>fixe</value>
12               <value>fax</value>
13             </choice>
14           </attribute>
15           <text/>
16         </element>
17       </zeroOrMore>
18     </element>
19   </oneOrMore>
20 </element>
```

Entiers et réels

articles.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <catalogue>
3    <article>
4      <description>Truc</description>
5      <prix>2.5</prix>
6      <quantité>14</quantité>
7    </article>
8    <article>
9      <description>Bidule</description>
10     <prix>17.5</prix>
11     <quantité>3</quantité>
12   </article>
13 </catalogue>
```

Contraintes :

- article : texte
- prix : réel (positif)
- quantité : entier (positif)

Une solution

articles.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="catalogue"
3           xmlns="http://relaxng.org/ns/structure/1.0"
4           datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
5    <oneOrMore>
6      <element name="article">
7        <element name="description"><text/></element>
8        <element name="prix">
9          <data type="float">
10           <param name="minExclusive">0</param>
11         </data>
12       </element>
13       <element name="quantité"><data type="positiveInteger"/></element>
14     </element>
15   </oneOrMore>
16 </element>
```

Support des *namespaces*

L'attribut `ns` d'une déclaration de *pattern* permet de préciser l'URI associé à l'élément ou à l'attribut défini :

adresse-ns.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="adresse"
3           xmlns="http://relaxng.org/ns/structure/1.0"
4           ns="http://apiacoa.org/teaching/xml/adresses">
5    <optional><element name="num"><text/></element></optional>
6    <choice>
7      <element name="rue"><empty/></element>
8      <element name="avenue"><empty/></element>
9      <element name="boulevard"><empty/></element>
10     <element name="place"><empty/></element>
11   </choice>
12   <text/>
13   <element name="code"><text/></element>
14   <element name="ville"><text/></element>
15 </element>
```

Documents

Valides :

une-adresse-ok.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <adresse xmlns="http://apiacoa.org/teaching/xml/adresses">
3   <num>15</num> <rue/> de belleville <code>75020</code>
4   <ville>Paris</ville>
5 </adresse>
```

une-adresse-ok-2.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <add:adresse xmlns:add="http://apiacoa.org/teaching/xml/adresses">
3   <add:num>15</add:num> <add:rue/> de belleville <add:code>75020</add:code>
4   <add:ville>Paris</add:ville>
5 </add:adresse>
```

Non valide :

une-adresse-non-ok.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <adresse>
3   <num>15</num> <rue/> de belleville <code>75020</code>
4   <ville>Paris</ville>
5 </adresse>
```

Modularité

- inclusion par l'élément `externalRef` (attribut `href`) qui agit comme un élément `ref` (fonctionne donc au niveau *pattern*)
- inclusion d'une grammaire entière par l'élément `include` (attribut `href`) :
  - mécanisme de fusion de définition (par alternative ou par fusion)
  - mécanisme de redéfinition
- exemple d'application : récupérer le *pattern* des adresses pour une liste de personnes

Le schéma combiné

personnes-add-2.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="liste"
3           xmlns="http://relaxng.org/ns/structure/1.0"
4           ns="http://apiacoa.org/teaching/xml/personnes">
5    <oneOrMore>
6      <element name="personne">
7        <element name="nom"><text/></element>
8        <element name="prénom"><text/></element>
9        <externalRef href="adresse-ns.rng"/>
10     </element>
11   </oneOrMore>
12 </element>
```

Exemple de fichier valide

liste-personnes-add.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <liste xmlns="http://apiacoa.org/teaching/xml/personnes"
3         xmlns:add="http://apiacoa.org/teaching/xml/adresses">
4    <personne>
5      <nom>Doe</nom>
6      <prénom>John</prénom>
7      <add:adresse>
8        <add:num>15</add:num> <add:rue/> de belleville
9        <add:code>75020</add:code>
10       <add:ville>Paris</add:ville>
11     </add:adresse>
12   </personne>
13 </liste>
```

Autre possibilité

liste-personnes-add-2.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <liste xmlns="http://apiacoa.org/teaching/xml/personnes">
3    <personne>
4      <nom>Doe</nom>
5      <prénom>John</prénom>
6      <adresse xmlns="http://apiacoa.org/teaching/xml/adresses">
7        <num>15</num> <rue/> de belleville
8        <code>75020</code>
9        <ville>Paris</ville>
10     </adresse>
11   </personne>
12 </liste>
```

SCHEMAS W3C

Les Schémas

Exemple de base : racine `a` contenant un nombre arbitraire de `b` vides.

DTD :

abstar.dtd

```
1 <!ELEMENT a (b*)>
2 <!ELEMENT b EMPTY>
```

Association :

test-abstar-dtd.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE a SYSTEM "abstar.dtd">
3 <a><b/><b/></a>
```

Schéma pour le même modèle

abstar.xsd

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  <xsd:element name="a">
4    <xsd:complexType>
5      <xsd:sequence>
6        <xsd:element name="b" minOccurs="0" maxOccurs="unbounded">
7          <xsd:complexType/>
8        </xsd:element>
9      </xsd:sequence>
10   </xsd:complexType>
11 </xsd:element>
12 </xsd:schema>
```

test-abstar-xsd.xml

```
1  <?xml version="1.0"?>
2  <a xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:noNamespaceSchemaLocation="abstar.xsd">
4  <b/><b/></a>
```

Schéma Relax NG pour le même modèle

abstar.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="a"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <zeroOrMore>
5      <element name="b">
6        <empty/>
7      </element>
8    </zeroOrMore>
9  </element>
```

test-abstar-rng.xml

```
1  <?xml version="1.0"?>
2  <a><b/><b/></a>
```

Les types

Les schémas sont basés sur la notion de type :

- chaque élément et chaque attribut possède un type
- approche objet : types de base et types défini par dérivation
- deux grandes catégories de types :

1. les types **simples** (`simpleType`) :

   chaînes de caractères, valeurs numériques, etc. Un élément d'un type simple ne peut ni contenir d'autres éléments ni avoir des attributs. Les attributs ont des types simples.

2. les types **complexes** (`complexType`) :

   tout le reste, en particulier les éléments contenant d'autres éléments et/ou des attributs.

Contenu d'un schéma

Un schéma :

- racine `schema`
- une suite de définition de types, d'éléments et d'attributs
- les descendants directs de la racine sont de niveau global (*top level*)
- structure assez souple :
  - identification et référence
  - types anonymes (définis localement)

Référence dans les définitions

Variante :

abstar-ref.xsd

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  <xsd:element name="a">
4    <xsd:complexType>
5      <xsd:sequence>
6        <xsd:element ref="b" minOccurs="0" maxOccurs="unbounded"/>
7      </xsd:sequence>
8    </xsd:complexType>
9  </xsd:element>
10 <xsd:element name="b">
11   <xsd:complexType/>
12 </xsd:element>
13 </xsd:schema>
```

Référence dans les définitions (2)

Variante :

abstar-ref2.xsd

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  <xsd:element name="a" type="TypeOfA"/>
4  <xsd:complexType name="TypeOfA">
5    <xsd:sequence>
6      <xsd:element ref="b" minOccurs="0" maxOccurs="unbounded"/>
7    </xsd:sequence>
8  </xsd:complexType>
9  <xsd:element name="b">
10   <xsd:complexType/>
11 </xsd:element>
12 </xsd:schema>
```

Un type sans nom est un type anonyme (!)

Définition des éléments

On définit un élément :

- au niveau global
- directement dans la définition d'un type complexe
- avec un élément `element` :
  - attribut `name` : nom de l'élément
  - attribut `type` : type de l'élément
  - attribut `fixed` : valeur de l'élément fixée et précisée
  - attribut `default` : valeur par défaut quand l'élément est présent mais avec un contenu vide
- si on ne donne pas un nom de type, on doit définir le type de l'élément directement comme descendant du noeud `element`

Attributs

Les attributs sont déclarés :

- au niveau global
- directement dans la définition d'un type complexe
- grâce à un élément `attribut` :
  - attribut `name` : nom de l'attribut
  - attribut `type` : type de l'attribut (obligatoirement un type simple, éventuellement obtenu par dérivation)
  - attribut `fixed` : valeur fixée
  - attribut `default` : valeur par défaut

Types simples

De nombreux types simples sont disponibles :

- chaînes de caractères (`string`), remplace `(#PCDATA)`
- types entiers (`byte`, `int`, etc.)
- types réels (`float`, `double`, etc.)
- dates et durées
- URI
- etc.

On peut restreindre chaque type grace à des *facets* (on obtient un type **dérivé**, toutjours simple) :

- valeur minimale ou/et maximale
- longueur (minimale, maximale, exacte)
- nombre de chiffres
- expression régulière
- énumération des valeurs possibles
- etc.

Exemple de types simples

- entier de l'intervalle $[0, 100]$

petit-entier.xsd

```
1  <xsd:simpleType name="petitEntier">
2    <xsd:restriction base="xsd:integer">
3      <xsd:minInclusive value="0"/>
4      <xsd:maxInclusive value="100"/>
5    </xsd:restriction>
6  </xsd:simpleType>
```

- numéro de téléphone à 10 chiffres

telephone.xsd

```
1  <xsd:simpleType name="telephone10">
2    <xsd:restriction base="xsd:string">
3      <xsd:pattern value="(\d{2} ){4}\d{2}"/>
4    </xsd:restriction>
5  </xsd:simpleType>
```

Types simples

Autres possibilités :

- un type liste à partir d'un type simple
- un type union à partir de plusieurs types simples
- les types simples restreints, les types liste et les types union sont toujours des types simples : on peut donc enchaîner les opérations de restriction, de mise en liste et d'union
- un type complexe à partir d'un type simple : ceci permet d'ajouter des attributs.

Exemple

telephone2.xsd

```
1  <xsd:complexType name="telephoneType">
2    <xsd:simpleContent>
3      <xsd:extension base="telephone10">
4        <xsd:attribute name="type" use="required">
5          <xsd:simpleType>
6            <xsd:restriction base="xsd:string">
7              <xsd:enumeration value="portable"/>
8              <xsd:enumeration value="fixe"/>
9              <xsd:enumeration value="fax"/>
10           </xsd:restriction>
11         </xsd:simpleType>
12       </xsd:attribute>
13     </xsd:extension>
14   </xsd:simpleContent>
15 </xsd:complexType>
```

Types complexes

La construction de base est la séquence :

- élément `sequence`
- le contenu indique les fils exigés pour le type grâce à des éléments `element` :
  - attribut `ref` : permet d'utiliser un élément déjà défini
  - attributs `minOccurs` et `maxOccurs` : nombre d'occurrence
  - on peut repréciser les valeurs par défaut
- contenu mixte : attribut `mixed="true"` pour l'élément `complexType` (attention plus strict que les DTDs)

On précise les attributs grâce à l'élément `attribut` :

- attribut `ref` : permet d'utiliser un attribut déjà défini
- attribut `use` : précise si l'attribut est `optional`, `prohibited` ou `required`
- on peut repréciser les valeurs par défaut

Exemple

email.xsd

```
1  <xsd:element name="email">
2    <xsd:complexType>
3      <xsd:sequence>
4        <xsd:element name="id" type="xsd:string"/>
5        <xsd:element name="domain">
6          <xsd:simpleType>
7            <xsd:restriction base="xsd:string">
8              <xsd:pattern
9                value="(\p{IsBasicLatin})+(\.(\p{IsBasicLatin})+)+"/>
10           </xsd:restriction>
11         </xsd:simpleType>
12       </xsd:element>
13     </xsd:sequence>
14   </xsd:complexType>
15 </xsd:element>
```

Exemple : `<email><id>John.Doe</id><domain>email.com</domain></email>`

Exemple

bookmark.xsd

```
1  <xsd:element name="bookmark">
2    <xsd:complexType>
3      <xsd:sequence>
4        <xsd:element name="title" type="xsd:string"
5         minOccurs="0" maxOccurs="1"/>
6      </xsd:sequence>
7      <xsd:attribute name="href" use="required" type="xsd:anyURI"/>
8    </xsd:complexType>
9  </xsd:element>
```

Exemple :

```
<bookmark href="http://www.w3.org"><title>W3C Home</title></bookmark>
```

Exemple de la liste de personnes

personnes.xsd

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  <xsd:element name="liste">
4    <xsd:complexType>
5      <xsd:sequence>
6        <xsd:element name="personne" minOccurs="1" maxOccurs="unbounded">
7          <xsd:complexType>
8            <xsd:sequence>
9              <xsd:element name="prénom" type="xsd:string"/>
10             <xsd:element name="nom" type="xsd:string"/>
11           </xsd:sequence>
12         </xsd:complexType>
13       </xsd:element>
14     </xsd:sequence>
15   </xsd:complexType>
16 </xsd:element>
17 </xsd:schema>
```

Autres fonctionnalités

Les schémas permettent :

- d'introduire des commentaires (`annotation`)
- de créer des groupes nommés d'éléments :
  - comme forme de macro (remplace les entités paramètres)
  - pour choisir entre plusieurs modèles de contenu pour un élément
  - pour relâcher légèrement les contraintes de contenu (zéro ou une occurence de chaque élément du groupe, dans un ordre quelconque)
- de créer des groupes nommés d'attributs (remplace les entités paramètres)

Autres fonctionnalités (2)

Les schémas permettent :

- de définir l'espace de nom créé (gestion fine et assez complexe)
- de construire des types par extension de types simples et par restriction de types complexes
- de contrôler les dérivations effectuées à partir d'un type
- de construire un schéma à partir d'autres schémas
- de demander l'unicité (locale) d'un attribut ou d'un élément (par exemple pour permettre des références croisées)
- etc.

Le standard est très riche et relativement complexe : c'est un véritable langage.