Modélisation et interopérabilité : Semaine 43, cours 6

Benoît Valiron <benoit.valiron@monoidal.net>

http://inf356.monoidal.net/

Aujourd'hui, validation Relax-NG avancée

Résumé

- Appel :

```
<element name=”a”>
 <text />
</element>

<attribute name=”a”>
 <text />
</attribute>
```

- Occurences :

<zeroOrMore>, <oneOrMore>, <optional>

- Combinaisons :

<group>, <choice>, <interleave>, <mixed>

- Appel :

```
element a {
 text
}

attribute a {
 text
}
```

- Occurences : * + ?
- Combinaisons : , | & mixed

Utilisation de (et) pour combinaisons complexes.

Patterns

- Centralisation des définitions
- Permet la récursion

```
<define name="elt-def">
 <element name="elt">
  <attribute name="att" />
  <text />
 </element>
</define>

<element name="parent">
 <attribute name="att2" />
 <ref name="elt-def" />
</element>
```

```
elt-def = element elt {
    attribute att { text },
    text
}

element parent {
 attribute att2 { text },
 elt-def
}
```

- La référence peur contenir n'importe quelle combinaison de patterns, y compris un appel à elle-même.

Élément racine

- Dans le cas d'utilisation de patterns:

<grammar> … </grammar> et <start> … </start>

```
<grammar>
 <define name=”pattern1”>
  ...
 </define>
 ...
 <start>
  <element name=”racine”>
   ...
  </element>
 </start>
</grammar>
```

```
pattern1 = ...
pattern2 = ...
...

start = element racine …
```

Exemple

```
<grammar>
<define name="atts_communs">
 <attribute name="lang"  />
 <attribute name="id"    />
 <attribute name="style" />
 <attribute name="class" />
</define>
<start>
<element name="span">
 <ref name="atts_communs" />
 <text />
</element>
</start>
</grammar>
```

Equivaut à

```
<element name="span">
 <attribute name="lang"  />
 <attribute name="id"    />
 <attribute name="style" />
 <attribute name="class" />
 <text />
</element>
```

```
atts_communs = (
 attribute lang  { text },
 attribute id    { text },
 attribute style { text },
 attribute class { text }
)

start = element span {
 atts_communs,
 text
}
```

Equivaut à

```
element span {
 ( attribute lang  { text },
   attribute id    { text },
   attribute style { text },
   attribute class { text }
 ),
 text
}
```

Exemple

```
<grammar>
  <define name=”elt_rec”>
   <element name=”elt”>
    <optional>
     <ref name=”elt_rec” />
    </optional>
   </element>
  </define>

  <start>
   <ref name=”elt_rec” />
  </start>
</grammar>
```

```
elt_rec =
 element elt { elt_rec ? }

start = elt_rec
```

- <elt></elt>
- <elt><elt></elt></elt>
- <elt><elt><elt></elt></elt></elt>
- <elt><elt><elt><elt></elt></elt></elt></elt>
- ...

Exemple

```
<grammar>
  <define name="rec_content">
    <zeroOrMore>
      <choice>
        <ref name="rec_i" />
        <ref name="rec_b" />
        <text/>
      </choice>
    </zeroOrMore>
  </define>

  <define name="rec_b">
    <element name="b">
      <ref name="rec_content" />
    </element>
  </define>

  <define name="rec_i">
    <element name="i">
      <ref name="rec_content" />
    </element>
  </define>

  <start>
    <element name="paragraph">
      <ref name="rec_content" />
    </element>
  </start>
</grammar>
```

```
rec_content =
  ( rec_b | rec_i | text )*

rec_i = element i { rec_content }
rec_b = element b { rec_content }

start =
  element paragraph { rec_content }
```

Exemple qui fait pas ce qu'on veut

```
<grammar>
  <define name="rec_content">
    <zeroOrMore>
      <mixed>
        <ref name="rec_i" />
        <ref name="rec_b" />
      </mixed>
    </zeroOrMore>
  </define>

  <define name="rec_b">
    <element name="b">
      <ref name="rec_content" />
    </element>
  </define>

  <define name="rec_i">
    <element name="i">
      <ref name="rec_content" />
    </element>
  </define>

  <start>
    <element name="paragraph">
      <ref name="rec_content" />
    </element>
  </start>
</grammar>
```

```
rec_content =
   mixed { rec_b, rec_i } *

rec_i = element i { rec_content }
rec_b = element b { rec_content }

start =
  element paragraph { rec_content }
```

Exemple où ça va pas

```
<grammar>
  <define name=”elt_rec”>
   <element name=”elt”>
    <ref name=”elt_rec” />
   </element>
  </define>

  <start>
   <ref name=”elt_rec” />
  </start>
</grammar>
```

```
elt_rec =
 element elt { elt_rec }

start = elt_rec
```

Exemple où ça va pas

```
<grammar>
  <define name="listatt">
   <attribut name="att" />
  </define>

  <start>
   <element name="elt">
    <ref name="listatt" />
    <attribut name="att" />
   </element>
  </start>
</grammar>
```

```
listatt =
 attribut att { text }


start = element elt {
 listatt,
 attribute att { text }
}
```

Traduction de DTDs

```
<!ELEMENT elt (a, (b | c))

<!ATTLIST elt att1 CDATA #REQUIRED
              att2 CDATA #IMPLIED
              att3 CDATA #IMPLIED>

<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
<!ELEMENT c EMPTY>
```

```
ref-elt = element elt {
  ref-elt-att,
  (ref-a, (ref-b | ref-c))
}

ref-elt-att = (
  attribute att1 { text },
  attribute att2 { text }?,
  attribute att3 { text }?
)

ref-a = element a { a-att, empty }
a-att = empty
ref-b = element b { b-att, empty }
b-att = empty
ref-c = element c { c-att, empty }
c-att = empty

start = ref-elt
```

Types simples

- Les seuls types natifs dans Relax NG sont les types `string` et `token`.
- Pour avoir plus de types, il faut faire appel à une bibliothèque de types ; par exemple, celle de XML Schema. E.g.

```
<element name=”elt”
    xmlsn=”http://relaxng.org/ns/structure/1.0”
    datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
 <data type=”nonNegativeInteger”>
</element>
```

Ou

```
datatypes xsd=”http://www.w3.org/2001/XMLSchema-datatypes”
element elt { xsd:nonNegativeInteger }
```

Librairie XMLSchema

- Chaines de caractères : en plus des type Relax NG,
  - `NMTOKEN` : comme pour les DTDs, pas d'espaces ni de ponctuation
  - `ID`, `IDREF`, `IDREFS` : comme pour les DTDs
  - `language` : une langue dans le code standard (ex: `en`, `en-US`, `fr`, `it`...)
- URIs :
  - `anyURI` : une adresse internet, un fichier, … Attention aux caractères accentués.
- Nombres et booléens :
  - `boolean` : true, false, 0 or 1
  - `decimal`, `integer`, `nonPositiveInteger`, `PositiveInteger`, `nonNegativeInteger`, `NegativeInteger`, `int` (32 bits), `short` (16 bits), `byte` (8 bits), …
- Date et heure :
  - `dateTime` : CCYY-MM-DDThh:mm:ss(timezone)
  - `date` : CCYY-MM-DD(timezone)
  - `time` : hh:mm:ss(timezone). Ex : 15:20:00 ou 18:00:15+02:00
  - `gYear` (ex. 2001), …

Exemple : une bibliothèque

```
<library>
 <book id=”n1”
       available=”yes”>
  <isbn>01-2345-6789</isbn>
  <title xml:lang=”fr”>
    Madame Bovary
  </title>
  <author>
   <name>Flaubert</name>
   <born>1821-12-12</born>
   <died>1880-05-08</died>
  </author>
 </book>
</library>
```

```
element library {
 element book {
  attribute id { xsd:ID } ,
  attribute available { xsd:boolean },
  element isbn { xsd:NMTOKEN },
  element title {
    attribute xml:lang { xsd:language },
    xsd:token
  },
  element author {
   element name { xsd:token },
   element born { xsd:date },
   element died { xsd:date }?
  }+
 }*
}
```

Valeur fixée

```
<element name=”isbn”>
 <value>012345567</value>
</element>
```

ou

```
element isbn { “01234567” }
```

Avec type :

```
<element
   name=”isbn”
   datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
 <value type=”NMTOKEN”>012345567</value>
</element>
```

ou

```
element isbn { xsd:NMTOKEN “01234567” }
```

Définition dépendante

```
element bibliothèque {
 element livre {
  attribute présent { xsd:boolean “true” },
  element titre { text },
  element emplacement { text }
 } |
 element livre {
  attribute présent { xsd:boolean “false” },
  element titre { text },
  element emprunteur {
   attribute carte_id { xsd:ID },
   element nom { text }
  }
 }
}
```

Définition dépendante

```
element bibliothèque {
 (element livre {
  attribute présent { xsd:boolean “true” },
  element titre { text },
  element emplacement { text }
 } |
 element livre {
  attribute présent { xsd:boolean “false” },
  element titre { text },
  element emprunteur {
   attribute carte_id { xsd:ID },
   element nom { text }
  }
 })*
}
```

Définition dépendante

```
element bibliothèque {
 element livre {
  (
    attribute présent { xsd:boolean “true” },
    element titre { text },
    element emplacement { text }
  ) |
  (
    attribute présent { xsd:boolean “false” },
    element titre { text },
    element emprunteur {
      attribute carte_id { xsd:ID },
      element nom { text }
    }
  )
 }*
}
```

Énumération

```
attribute état { “présent” | “absent” | “ne sais pas” }
```

ou

```
<attribute name=”état”>
  <choice>
    <value>présent</value>
    <value>absent</value>
    <value>ne sais pas</value>
  </choice>
</attribute>
```

Par défaut, le type utilisé est `token`

Énumération : avec types

```
attribute premier {
  xsd:int “2” | xsd:int “3” |
  xsd:int “5” | xsd:int “7”
}
```

ou

```
<attribute name=”premier”
     datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
  <choice>
    <value type=”int”>2</value>
    <value type=”int”>3</value>
    <value type=”int”>5</value>
    <value type=”int”>7</value>
  </choice>
</attribute>
```

Énumération et types variés

```
attribute nombres {
  “trop petit” |
  xsd:integer “1” |
  xsd:nonNegativeInteger “2” |
  xsd:string “trop grand”
}
```

ou

```
<attribute name=”nombre”
     datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
  <choice>
    <value>trop petit</value>
    <value type=”int”>1</value>
    <value type=”nonNegativeInteger”>2</value>
    <value type=”string”>trop grand</value>
  </choice>
</attribute>
```

Listes

```
element valeurs {
  list {xsd:boolean, xsd:boolean, xsd:boolean}
}
```

ou

```
<element name=”valeurs”>
  <list>
    <data type=”boolean” />
    <data type=”boolean” />
    <data type=”boolean” />
  </list>
</element>
```

Attention :

```
element valeurs {
  list {xsd:int}
}
```

Est une liste d'un seul élément

Exemple :

```
<valeurs>0 1 false</valeurs>      VALIDE

<valeurs>23 false true</valeurs> NON VALIDE
```

Listes, toujours

```
element nombres {
  list {xsd:unsignedBytes*}
}
```

ou

```
<element name=”nombres”>
 <list>
  <zeroOrMore>
   <data type=”unsignedByte” />
  </zeroOrMore>
 </list>
</element>
```

Exemple :

```
<valeurs>1 2 43 55 0068 256</valeurs>
```
VALIDE

```
<valeurs>-1 22 34 1</valeurs>
```
NON VALIDE

Listes et énumérations

```
element nombres {
  list {(xsd:int “1” | xsd:int “10” | xsd:int “100”)+}
}
```

ou

```
<element name=”nombres”>
 <list>
  <oneOrMore>
   <choice>
    <value type=”int”>1</value>
    <value type=”int”>10</value>
    <value type=”int”>100</value>
   </choice>
  </oneOrMore>
 </list>
</element>
```

Encore plus complexe

```
element taille {
  list {xsd:nonNegativeInteger, (“cm” | “m” | “mm”)}
}
```

ou

```
<element name=”taille”>
 <list>
  <data type=”nonNegativeInteger” />
  <choice>
   <value>cm</value>
   <value>m</value>
   <value>mm</value>
  </choice>
 </list>
</element>
```

Exemple :

```
<taille>1 cm</taille>

<taille>23 m </taille>
```

Facettes des types XMLSchema

- Ce sont des restrictions : tailles des chaines de caractères, propriétés des entiers, etc.
- Elles sont définies dans la librairie.
- Appel :

```
<element name=”elt”>
 <data type=”decimal”>
  <param name=”maxInclusive”>56.7</param>
 </data>
</element>
```

```
element elt {
  xsd:decimal { maxInclusive=”56.7” }
}
```

Liste de facettes

- `length`, `maxLenth`, `minLength` : longueur de chaine de caractères
- `maxExclusive`, `minExclusive`, `maxInclusive`, `minInclusive`
- `totalDigits` : s'applique à `decimal`, `integer`
- `pattern` : expression régulière sur la chaine de caractères donnée.

Exemples

Pas de virgule

```
attribute n {
  xsd:int { pattern=”1|2” }
}
```

n=”1” ~~n=”001”~~
n=”2”

```
attribute n {
  xsd:int “1”
}
```

n=”1”
n=”01”
n=”001”

```
element date {
 xsd:date {
  minInclusive=”2000-01-01”
  maxInclusive=”2000-04-30”
 }
}
```

```
attribute password {
 xsd:string {
  length=”6”
 }
}
```

Espaces de noms

- Avec Relax NG, la gestion des espaces de noms est simple et intégré au système.
  - Espace de nom par défault :

```
<element ns=”espace-de-noms” name=”elt”>
 ...
</element>
```

```
default namespace = “espace-de-noms”
element elt {
 ...
}
```

Attention : comme d'habitude, les attributs ne prennent pas l'espace de nom par défaut.

  - Sinon : avec `xmlns:...`

```
<element xmlns:pre=”espace-de-noms” name=”pre:elt”>
 ...
</element>
```

```
namespace pre = “espace-de-noms”
element pre:elt {
 ...
}
```

Exemple

```
<element name=”RDF”
     ns=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
     xmlns:dc=”http://purl.org/dc/elements/1.1/”
     datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
 <element name=”Description”>
  <attribute name=”about”><data type=”anyURI” /></attribute>
  <oneOrMore>
   <choice>
    <element name=”dc:creator”><text/></element>
    <element name=”dc:date”><data type=”date”/></element>
    <element name=”dc:description”><text/></element>
    <element name=”dc:format”><text/></element>
   <choice>
  </oneOrMore>
 </attribute>
</element>
```

Exemple

```
<element name=”rdf:RDF”
     xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
     xmlns:dc=”http://purl.org/dc/elements/1.1/”
     datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
 <element name=”rdf:Description”>
  <attribute name=”about”><data type=”anyURI” /></attribute>
  <oneOrMore>
   <choice>
    <element name=”dc:creator”><text/></element>
    <element name=”dc:date”><data type=”date”/></element>
    <element name=”dc:description”><text/></element>
    <element name=”dc:format”><text/></element>
   <choice>
  </oneOrMore>
 </attribute>
</element>
```

Exemple

```
<element name=”rdf:RDF”
     xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
     ns=”http://purl.org/dc/elements/1.1/”
     datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
 <element name=”rdf:Description”>
  <attribute name=”about”><data type=”anyURI” /></attribute>
  <oneOrMore>
   <choice>
    <element name=”creator”><text/></element>
    <element name=”date”><data type=”date”/></element>
    <element name=”description”><text/></element>
    <element name=”format”><text/></element>
   <choice>
  </oneOrMore>
 </attribute>
</element>
```

Exemple

```
<RDF xmlns=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”>
 <Description xmlns:dub=”http://purl.org/dc/elements/1.1/”>
  <pub:creator>Benoît Valiron</pub:creator>
  <pub:date>2009-10-21</pub:date>
 </Description>
</RDF>

<qwerty:RDF
    xmlns:qwerty=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
    xmlns=”http://purl.org/dc/elements/1.1/”>
 <qwerty:Description>
  <creator>Benoît Valiron</creator>
  <date>2009-10-21</date>
 </qwerty:Description>
</qwerty:RDF>
```

Exemple (reprise du TD 4)

ville.xml

```
<ville xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
  <nom>Grenoble</nom>
  <geo:lat>45.196349</geo:lat>
  <geo:long>5.73226</geo:long>
</ville>
```

ville.rnc

```
namespace geo = “http://www.w3.org/2003/01/geo/wgs84_pos#”

element ville {
  element nom { xsd:token },
  element geo:lat { xsd:decimal },
  element geo:long { xsd:decimal }
}
```

Exemple (reprise du TD 4)

ville.xml

```
<ville xmlns:g="http://www.w3.org/2003/01/geo/wgs84_pos#">
  <nom>Grenoble</nom>
  <g:lat>45.196349</g:lat>
  <g:long>5.73226</g:long>
</ville>
```

ville.rnc

```
namespace geo = “http://www.w3.org/2003/01/geo/wgs84_pos#”

element ville {
  element nom { xsd:token },
  element geo:lat { xsd:decimal },
  element geo:long { xsd:decimal }
}
```

Exemple (reprise du TD 4)

ville.xml

```
<ville>
  <nom>Grenoble</nom>
  <lat xmlns=”http://www.w3.org/2003/01/geo/wgs84_pos#”>
    45.196349
  </lat>
  <long xmlns=”http://www.w3.org/2003/01/geo/wgs84_pos#”>
    5.73226
  </long>
</ville>
```

ville.rnc

```
namespace geo = “http://www.w3.org/2003/01/geo/wgs84_pos#”

element ville {
  element nom { xsd:token },
  element geo:lat { xsd:decimal },
  element geo:long { xsd:decimal }
}
```

Annotations

- Idée : Ajouter de l'information à un schéma
  - Documentation
  - Ajout de fonctionalité
- Traités par les outils comprenant l'ajout
- Laissé de côté par les autres.
  - La documentation n'apporte rien
  - Intégration d'autre technique de validations qui seront ignorés si non comprises par un outils donné.

Technique

- Syntaxe XML : simplement l'utilisation d'un espace de nom autre que celui de Relax NG (voire pas d'espace de nom du tout)

```
<grammar xmlns=”http://relaxng.org/ns/structure/1.0"
         xmlns:dc=”http://purl.org/dc/elements/1.1/”>
  <dc:creator>Merlin</dc:creator>
  <dc:description>Un Schéma magique !</dc:description>
  <start>
    <element name=”elt”>
      <text />
    </element>
  </start>
</grammar>
```

- Syntaxe compacte :

```
default namespace = “http://relaxng.org/ns/structure/1.0"
namespace dc = ”http://purl.org/dc/elements/1.1/”

dc:creator [ “Merlin” ]
dc:description [ “Un Schéma magique !” ]
start = element elt { text }
```

Documentation d'element

- Syntaxe XML :

```
<grammar xmlns=”http://relaxng.org/ns/structure/1.0"
         xmlns:dc=”http://purl.org/dc/elements/1.1/”>
  <dc:creator>Merlin</dc:creator>
  <start>
  <element name=”elt”>
    <dc:description>
        Ceci est la racine
    </dc:description>
    <text />
  </element>
  </start>
</grammar>
```

- Syntaxe compacte :

```
default namespace = “http://relaxng.org/ns/structure/1.0"
namespace dc = ”http://purl.org/dc/elements/1.1/”

dc:creator [ “Merlin” ]
start = (
  [ dc:description [ “Ceci est la racine !” ] ]
  element elt {
    text
  }
)
```

Exemple

```
namespace dc  = “http://purl.org/dc/elements/1.1/”
namespace geo = “http://www.w3.org/2003/01/geo/wgs84_pos#”

dc:creator [ “GeoLocatorSoftware“ ]
dc:description [ “Relevé GPS d'une ville” ]

start = (
  [ dc:description [ “Tout est ici” ] ]
  element ville {
    [ dc:description [ “La ville en question” ]]
    element nom { xsd:token },
    [ dc:description [ “Latitude de la ville” ]]
    element geo:lat { xsd:decimal },
    [ dc:description [ “Longitude de la ville” ]]
    element geo:long { xsd:decimal }
  }
)
```

Exemple

```
<grammar
     xmlns:geo=”http://www.w3.org/2003/01/geo/wgs84_pos#”
     xmlns:dc=”http://purl.org/dc/elements/1.1/”
     datatypeLibrary=”http://www.w3.org/2001/XMLSchema-datatypes”>
<dc:creator>Geo-Locator-Software</dc:creator>
<dc:description>Relevé GPS d'une ville</dc:description>
<start>
<element name=”ville”>
  <dc:description>Tout est ici</dc:description>
  <element name=”nom”>
    <dc:description>La ville en question</dc:description>
    <data type=”xsd:token”/>
  </element>
  <element name=”geo:lat”>
    <dc:description>Latitude de la ville</dc:description>
    <data type=”xsd:decimal”/>
  </element>
  <element name=”geo:long”>
    <dc:description>Longitude de la ville</dc:description>
    <data type=”xsd:decimal” />
  </element>
</element>
</start>
</grammar>
```

Documentation de Schema

- Aussi simplement avec des commentaires...
- Format XML :

```
<element name=”elt”>
  <!-- ceci est un bel élément →
  <text />
</element>
```

- Format compact :

```
element elt {
  # ceci est un bel élément
  text
}
```

Utilisations avancées

- Combinaison de Relax-NG et d'autre validateurs:

```
<element name="Root" xmlns="http://relaxng.org/ns/structure/1.0">
   <sch:pattern name="Test constraints on the Root element"
                xmlns:sch="http://www.ascc.net/xml/schematron">
    <sch:rule context="Root">
      <sch:assert test="test-condition">
         Error message when the assertion condition is broken...
      </sch:assert>
    </sch:rule>
   </sch:pattern>
   <text/>
</element>
```

- Valeurs par défaut d'attributs (simulation de DTD)

```
<grammar xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0”
         xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="elt">
      <optional>
        <attribute name="att"
           a:defaultValue="valeur"/>
      </optional>
      <empty/>
    </element>
  </start>
</grammar>
```

```
start =
  element elt {
    [ a:defaultValue = "valeur" ]
    attribute att { text }?,
    empty
  }
```

Exemple : les sitemaps

- Le moteur de recherche google propose un service aux webmasters pour l'indexation : les sitemaps. Il s'agit d'un format XML pour donner des informations succintes au robot d'indexation sur les pages qu'il peut rencontrer. Le format est le suivant (pris sur wikipedia) :
  - Élément <urlset>. Obligatoire. Racine du document.
  - Élément <url>. Obligatoire. Élément parent pour chaque entrée. Les éléments restant sont tous fils de cet élément.
  - Élément <loc>. Obligatoire. Contient l'adresse internet d'une page, incluant le protocole (http:// ou https://). Doit faire au maximum 2048 caractères de long.
  - Élément <lastmod>. Facultatif. La date de dernière modification du fichier, en format ISO ou plus simplement YYYY-MM-DD.
  - Élément <changefreq>. Facultatif. Fréquence à laquelle la page est modifiée en générale : always, hourly, daily, weekly, monthly, yearly, never.
  - Élément <priority>. Facultatif. L'importance relative de cette page par rapport aux autres. Valeur entre 0.0 et 1.0, valeur par défaut de 0.5.