Les langages de schéma XML

Stéphane Bortzmeyer
AFNIC
bortzmeyer@nic.fr

14 février 2006

Exposé libre

XML n'est pas un langage

C'est un ensemble de règles de syntaxe de bas niveau. La syntaxe de haut niveau et la sémantique, la définition du langage, est ailleurs.

Exemples de langages XML :

- Docbook
- Atom (format de syndication, RFC 4287)
- EPP (protocole mais aussi langage, pour définir les éléments XML échangés)
- IRIS (idem)

Schéma

Chaque langage est décrit par un schéma.

Ce schéma peut être écrit dans différents langages.

- le langage originel, DTD (utilisé par le RFC 2629, par Docbook jusqu'à la version 4, l'actuelle),
- Examplotron, qui permet de créer un schéma à partir d'un document exemple,
- les W3C Schemas, souvent appelés XSD (utilisés par Wikipedia, par SOAP, ou bien pour EPP et IRIS),
- RelaxNG, le petit dernier (utilisé par OpenDocument, Docbook à partir de la version 5, Atom, RFC-editor queue, etc).

Différentes sortes de langage

- Langage à base de **règles** comme Schematron. “Ce qui doit être vrai”
- Langage à base de **grammaire**. “Ce qui est” :
  - Langage de description comme W3C Schema ou DTD. “Ce qu'il y a dans un élément.”
  - Langage de motifs comme RelaxNG. “Ce que doit respecter l'élément”

Les différences sont subtiles mais peuvent faire la différence. Certaines constructions ne sont **pas** expressibles en W3C Schema ou en DTD.

Rappel XML

XML est hiérarchique.

```
<foo id="exemple">
   <bar/>
</foo>
```

Le tout est un **document** XML, ce qui est entre les étiquettes `<foo>` et `</foo>` est un **élement**, qui est à la racine du document et dont le **contenu** est un autre élement.

`<bar/>` est un élément vide, **fils** de l'élement `<foo>`.

“id” est un **attribut**.

Bien formé et valide

Tout document XML est **bien formé**. Sinon, c'est une erreur.

```
<!-- Pas bien forme -->
<foo><bar/></fou>
```

```
<!-- Pas bien forme -->
<foo><baz></foo></baz>
```


Validité

Certains documents XML sont **valides**, selon un **schéma** écrit dans un langage de schéma comme DTD ou RelaxNG.

```
<!-- Bien forme mais pas valide selon la DTD Docbook -->
<article>Texte</article>
```

Après le fait d'être **bien formé** et la **validité**, il y a le respect des **règles métier** (tout ne peut pas être mis dans le schéma).

Les *namespaces*

Si on veut mélanger des éléments de plusieurs vocabulaires.

```
<afnic:domain holder='AR41-NIC'>
   <afnic:name>example.fr</afnic:name>
   <afnic:active/>
   <inpi:mark final='1'>123456789</inpi:mark>
</afnic:domain>
```

Les préfixes des espaces de noms (ici, “afnic” et “inpi”) sont définis par un URI (RFC 3305). C'est l'URI qui compte, **pas** le préfixe :

```
<afnic:domain holder='AR41-NIC'
         xmlns:afnic="http://www.afnic.fr/Confiance" >
   <afnic:name>example.fr</afnic:name>
   <afnic:active/>
   <inpi:mark final='1' xmlns:inpi="urn:INPI">123456789</inpi:mark>
</afnic:domain>
```


Pourquoi un langage de schéma ?

Avoir un schéma pour ses documents XML n'est **pas** obligatoire.

Mais cela permet :

- de les valider, donc de simplifier l'application qui les traite,
- et de guider le processus d'édition.

Caractéristiques du langage

Il y a deux choses à décrire dans un schéma :

1. La **structure**, c'est-à-dire la combinaison des éléments (le fait que tout `<article>` doive avoir un `<title>` par exemple). C'est le rôle d'une grammaire.
2. Le contenu des éléments, autrement dit leur **type** (le fait que `<date>` doive contenir un truc de la forme YYYY-MM-DD par exemple ou bien que `<salary>` doive être un entier positif).

Tous les langages de schéma séparent ces deux aspects.

Ambigüité et non-déterminisme

- L'ambigüité est le fait que deux alternatives de la grammaire puissent valider un document.
  Par exemple, la grammaire `"a" / "b"* "a"` est ambigüe car la lettre "a" peut être validée par les deux termes de l'alternative.
  Pas gênante pour valider mais plus embêtant pour l'édition guidée.
- Le non-déterminisme (l'ambigüité implique le non-déterminisme) est le fait que l'analyseur doive parfois regarder en avant pour savoir quelle branche de la grammaire prendre.
  Ainsi, `"a" / "a" "b"` n'est pas ambigüe mais est non-déterministe.

Exemple de langage XML

Pour les exemples, nous prendrons l'exemple suivant :

1. Un registre de noms de domaines est régi par la loi du 9 juillet 2004 qui dit que « En cas de cessation de l'activité de ces organismes, l'État dispose du droit d'usage de la base de données des noms de domaine qu'ils géraient. ». Il faut donc un dispositif de **séquestre**, pas de simple sauvegarde. On choisit XML.
2. On va donc créer un schéma pour décrire la base de données du registre, avec des <domain> et des <contact>.

Un exemple de zone

```
<zone name="fx">
   <domain>
      <name>foobar.fx</name>
      <holder>AB1-FXNIC</holder>
      <created>2006-01-16</created>
   </domain>
   <contact publish="false">
     <name>Bortzmeyer</name><firstname>Stephane</firstname>
     <handle>SB1-FXNIC</handle>
     <phone>+33 1 39 30 83 46</phone>
     <email>foo@bar</email>
   </contact>
</zone>
```

DTD, l'ancêtre et ses limites

Le langage original, issu de SGML.

- Syntaxe pas XML (avantage ou inconvénient ?)
- Uniquement la structure, pas les types
- Contraintes d'intégrité (ID/IDREF)

```
<!ELEMENT domain (name,nameservers?,holder,tech?,admin?,created)>

<!ELEMENT contact (name,firstname?,handle,address*,city?,country?,phone?,ema
<!ATTLIST contact publish NMTOKEN "false">

<!ELEMENT name (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT handle (#PCDATA)>
<!ELEMENT holder (#PCDATA)>
```


Ce qu'on ne peut pas faire avec la DTD

- Spécifier la syntaxe d'une date ou d'un booléen comme publish
- Spécifier des cardinalités autre que 0, 1 ou N
- Spécifier des contraintes d'intégrité avec les éléments (mais on peut avec les attributs)
- Ne **pas** spécifier d'ordre sur les éléments (<phone> et <email>)

Alors, DTD ou pas DTD ?

Le Cobol des langages de schéma

Tout le monde déteste et sait que c'est archaïque mais elles sont très répandues et beaucoup d'outils existent.

Examplotron

Modéliser commence souvent par un exemple

Les sessions de *brainstorming* travaillent en général mieux avec des exemples qu'avec des schémas.
Le principe est donc simple : un schéma est un document d'exemple comme celui vu plus haut.

Principale faiblesse : comment savoir si un élement est optionnel ou pas ?

On enrichit donc le document avec des annotations.

Examplotron avec annotations

```
... xmlns:eg="http://examplotron.org/0/" ...
  <contact publish="true" >
      <name>Renard</name><firstname>Annie</firstname>
      <handle>AR41-FXNIC</handle>
      <email>ar@nic.fx</email>
      <!-- Numero de telephone optionnel -->
      <phone eg:occurs="?">+33 1 39 30 00 41</phone>
   </contact>
```

Examplotron trouve tout seul les types mais on peut les forcer le cas échéant.

Mise en œuvre d'Examplotron

Une feuille de style XSLT traduit le document exemple en schéma RelaxNG.

On peut donc utiliser n'importe quel outil RelaxNG.

W3C Schemas, l'officiel

W3C Schemas, le langage d'EPP et d'IRIS.

Normalisé par le W3C. Bénéficie du meilleur *mindshare* : pour les lecteurs de 01, c'est un synonyme de schéma.

Syntaxe XML.

Un schéma W3C

```
<schema:element name="domain">
  <schema:complexType>
    <schema:sequence>
      <schema:element name="name" type="schema:string"/>
      <schema:element name="holder" type="schema:IDREF"/>
      <schema:element name="created" type="schema:date"/>
    </schema:sequence>
  </schema:complexType>
</schema:element>
 <schema:element name="contact">
  <schema:complexType>
    <schema:sequence>
      <schema:element name="name" type="schema:string"/>
      <schema:element name="firstname" type="schema:string"/>
      <schema:element name="email"  type="email-address" minOccurs="0"/>
      <schema:element name="phone"  type="schema:string" minOccurs="0"/>
    </schema:sequence>
    <schema:attribute name="publish" type="schema:boolean"/>
  </schema:complexType>
</schema:element>
```

Exemple IRIS

RFC 3982 :

```
<complexType
  name="domainType">
  <complexContent>
    <extension
      base="iris:resultType">
      <sequence>
        <element name="domainName"  type="token" />
        <element
          name="nameServer"
          type="iris:entityType"
          minOccurs="0"
          maxOccurs="unbounded" />
        <element
          name="registrant"
          type="iris:entityType"
          minOccurs="0"
          maxOccurs="1" />
        <element name="status"       minOccurs="0"                maxOc
          <complexType>
                <element
                  name="reservedDelegation"
```


W3C Schemas ou pas

Très verbeux, en partie en raison de la syntaxe XML : l'éditer à la main est difficile.

Paradoxalement peu d'outils, surtout en logiciel libre.

Des restrictions pénibles dans le langage.

RelaxNG, le *challenger*

http://www.relaxng.org/

Le langage de Docbook et d'Atom. Préféré par Oasis.

Deux syntaxes, XML et compacte.

Ne spécifie que la structure et compte sur une bibliothèques de types pour le contenu.

```
valid_domain_name =
      xsd:string {pattern = "[A-Za-z0-9\-\.]+"}
domain = element domain
            {domain_name & holder & created}
contact = element contact
            {publish?, (firstname & name & handle & email? & phone?)}
domain_name = element name
                  {valid_domain_name}
created = element created {xsd:date}
publish = attribute publish {xsd:boolean}
```


RelaxNG, exemple

Extrait d'Atom

```
element atom:entry {
      atomCommonAttributes,
      (atomAuthor*
       & atomId
       & atomRights?
       & atomTitle
       ...
   }
atomAuthor = element atom:author { atomPersonConstruct }
atomPersonConstruct =
   atomCommonAttributes,
   (element atom:name { text }
    & element atom:uri { atomUri }?
    & element atom:email { atomEmailAddress }?
    ...)
```

Le cas particulier du Schematron

`http://www.schematron.com/`

Schematron est un langage d'**assertions** sur les documents. Par exemple "tout élement <domain> doit avoir au moins deux éléments <nameserver>". Ces assertions sont exprimées en Xpath.

Il peut être utilisé seul ou bien en conjonction avec un autre langage de schémas.

Exemple Schematron

```
<?xml version="1.0"?>
<schema xmlns="http://www.ascc.net/xml/schematron" xmlns:axsl="http://www.w3
  <pattern>
    <rule context="foo">
      <assert test="bar">
         We need at least one bar
      </assert>
      <assert test="@id">
         We need the ID of the foo
      </assert>
    </rule>
    <rule context="bar">
      <assert test="count(descendant::*) = 0">
          bar must not have subelements
      </assert>
    </rule>
  </pattern>
</schema>
```

Combiner Schematron et Relax-NG

Ici, pour exprimer une contrainte d'intégrité (le *handle* du titulaire doit exister).

```
domain =
  element domain { domain_name
    & holder
       >> sch:pattern [
            name = "Integrity reference for holder"
            sch:rule [ context = "domain"
              sch:assert [test = "/zone/contact[handle=current()/holder]"
              "The " sch:name [] sch:value-of [select = "name"] " has an inval
    & created
  }
```


Bibliothèque de types

Certains langages de schéma comme RelaxNG ne gèrent que la structure.

Pour le type (le contenu des éléments), il faut un autre langage.

On peut utiliser une bibliothèque de types existantes comme celle des W3C Schema.

Ou bien avoir un langage d'écriture de types comme DTLL `http://www.jenitennison.com/datatypes/DTLL.html` ou comme rnv, avec qui on peut créer ses propres types en Scheme :

```
(define addr-spec-regex
(let* (
    (atom "[a-zA-Z0-9!#$%&'*+\\-/=?\\^_`{|}~]+")
    (person "\"([^\"\\\\]|\\\\.)\"")
    (location "\\[([^\\[\\]\\\\]|\\\\.)*\\]")
    (domain (string-append atom "(\\." atom ")*")))
  (string-append
    "(" domain "|" person ")"
    "@"
    "(" domain "|" location ")")))
```

Ouverture des schémas

Il est souvent souhaitable de modifier un schéma, de l'étendre ou bien de le réutiliser.

L'ouverture d'un schéma (sa capacité à être modifié, étendu ou réutilisé) ne dépend pas que du langage mais aussi de l'auteur.

Exemple de schéma rigide en RelaxNG :

```
foo = element foo {element bar {text}}
```

Exemple de schéma ouvert en RelaxNG :

```
bar = element bar {text}
foo = element foo {bar}
```

<bar> peut maintenant être utilisé seul.

Ouverture de notre schéma

Le schéma original :

```
domain = element domain {domain_name & holder & tech & created}
tech = tech_contact # Exactly one
tech_contact = element tech {xsd:NMTOKEN}
```

Le schéma étendu pour admettre plusieurs contacts techniques :

```
include "../domain-escrow/relaxng-schema.rnc"
tech &= tech_contact* # Merge with previous definition
```

Ouvrir, autre exemple

```
domain = element domain {domain_items}
domain_items = domain_name & holder & tech & created
```

Ainsi, on peut ajouter des attributs ou des éléments à <domain> sans changer le schéma :

```
uname = element uname {text} # Unicode name of the domain
domain_items &= uname?
```


namespaces

Il n'y a pas encore de mécanisme de schéma multi-*namespaces*. Exemple : un flux Atom qui contient de l'XHTML. Il n'y a pas de moyen propre de valider le contenu XHTML selon le schéma de XHTML.

Une proposition : NRL (*Namespace Routing Language*)
`http://www.thaiopensource.com/relaxng/nrl.html`

```
<rules xmlns="http://www.thaiopensource.com/validate/nrl">
  <namespace ns="http://www.w3.org/2005/Atom">
    <validate schema="atom.rnc"/>
  </namespace>
  <namespace ns="http://www.w3.org/1999/xhtml">
    <validate schema="xhtml.rng"/>
  </namespace>
</rules>
```

D'un langage à l'autre

trang `http://www.thaiopensource.com/relaxng/trang.html` convertit d'un langage de schéma dans l'autre.

```
%.dtd: %.rnc
        trang -Irnc -Odtd $< $@

%.xsd: %.rng
        trang -Irng -Oxsd $< $@
```


Outils de validation

Ligne de commande ou bibliothèque, le choix est vaste. Il existe même des *appliances* (`http://www.datapower.com/products/xa35.html` ou `http://www.sarvega.com/xml-processing.html`).

Attention : certains font souvent des faux négatifs (documents acceptés à tort).

xmllint

Ce programme fait partie de la **libxml2** de Gnome, écrite par Daniel Veillard.

Mais il peut s'utiliser sans Gnome.

Il sait valider contre une DTD, un schéma RelaxNG (des bogues avec les *interleaves*) ou un schéma W3C (tout n'est pas implémenté).

```
xmllint --noout --dtdvalid dtd-schema.dtd example.xml
xmllint --noout --relaxng relaxng-schema.rng example.xml
xmllint --noout --schema w3c-schema.xsd example.xml
```


jing

jing `http://www.thaiopensource.com/relaxng/jing.html` est écrit en Java et donc peu portable.

Ne connait que RelaxNG.

```
java -jar /local/lib/jing.jar relaxng-schema.rng example.xml
```

rnv

Écrit en C `http://www.davidashen.net/rnv.html`

Ne connait que RelaxNG/compact.

```
rnv relaxng-schema.rnc example.xml
```


Outils d'édition guidée

Édition guidée : complétion des *tags*, affichage des choix possibles, mode *outline*, etc.

nxml mode `http://www.thaiopensource.com/download/` est un mode Emacs. Le schéma doit être en RelaxNG/compact.

psgml `http://www.lysator.liu.se/~lenst/about_psgml/` est un mode Emacs pour SGML et qui marche donc pour XML. DTD seulement.

Premier exercice

Termes interdits

Modéliser en Examplotron une liste de termes interdits

1. Quels éléments ?
2. Quel type ?

Deuxième exercice

Listes de domaines pour lesquels on est secondaire

Modéliser en RelaxNG une liste de domaines dont on assure le DNS secondaire

1. Quels éléments ?
2. Quel type ?