DTD et schémas

Fabrice Rossi

26 janvier 2003

1 Instructions spécifiques au CRIO UNIX

Au début de chaque séance :

1. dans chaque terminal, utilisez tcsh (en tapant `tcsh`)
2. effectuez les réglages nécessaires au bon fonctionnement des différents programmes utilisés, en tapant `source /home/perm/ufrmd/rossi/xml/env.csh`

On valide un document en utilisant les commandes suivantes :
- `validate fichier` pour valider le `fichier` par rapport à une DTD ou un schéma W3C
- `validate-ng schéma fichier` pour valider le `fichier` par rapport à un `schéma` RELAX NG

Dans les deux cas, le programme affiche uniquement des messages d'erreur. L'absence d'affichage indique donc que le fichier étudié est valide.

Pour l'exercice 3.3, on utilisera les commandes suivantes :
- `validate-ng-type schéma fichier jar` qui valide le `fichier` par rapport à un `schéma` RELAX NG en utilisant la bibliothèque de types contenue dans le `jar`
- `compile-ng fichiers` qui compile un ensemble de fichiers en donnant accès aux interfaces et classes nécessaires à l'implantation d'une bibliothèque de types

2 DTD

Exercice 2.1

On souhaite représenter un carnet d'adresses en XML. Pour chaque entrée du carnet, on veut conserver les informations suivantes :
- nom et prénom de la personne
- adresse (éventuellement en dehors de France)
- numéro de téléphone (éventuellement plusieurs)
- adresse *mail* (éventuellement plusieurs)
- date d'anniversaire

Questions :

1. Écrire une DTD pour le carnet d'adresses.
2. Écrire un fichier valide pour cette DTD, comportant au moins deux entrées et faisant apparaître toutes les possibilités de la DTD (c'est-à-dire toutes les valeurs possibles pour les attributs et tous les éléments et attributs optionnels).

Exercice 2.2

On souhaite donner une version plus professionnelle du carnet d'adresses de l'exercice 2.1. Pour chaque entrée du carnet, on conserve les informations suivantes :
- nom et prénom de la personne
- numéro de téléphone (éventuellement plusieurs)
- adresse *mail* (éventuellement plusieurs)
- service

– entreprise (avec le site)

L'adresse de chaque contact est celle de son entreprise. Le fichier XML doit donc contenir, pour chaque entreprise, les informations suivantes :

– nom
– une liste de sites, avec pour chaque site :
  – nom du site
  – adresse du site

Pour éviter la redondance dans le fichier, le lien entre une personne et le site de l'entreprise pour laquelle elle travaille est obtenu grâce à une référence croisée.

Questions :

1. Écrire une DTD pour la nouvelle version du carnet.
2. Écrire un fichier valide pour cette DTD, comportant au moins une entreprise à deux sites et au moins deux personnes (une pour chaque site).

3 Schémas RELAX NG

Exercice 3.1

On souhaite représenter la structure d'une entreprise sous forme d'un fichier XML. Ce fichier sera organisé sous la forme d'une liste d'employés. Pour chaque employé, on indique :

– son nom et son prénom
– son bureau
– son titre (éventuellement, comme par exemple "directeur commercial")
– la liste de ses subordonnés

Comme dans l'exercice 2.2, on souhaite au maximum éviter la redondance, au moyen de références croisées.

Questions :

1. Proposer une DTD pour la structure d'entreprise.
2. Pour utiliser les types de contenu ID, IDREF et IDREFS avec RELAX NG, il faut passer par la bibliothèque de types `http://relaxng.org/ns/compatibility/datatypes/1.0`. Grâce à cette bibliothèque, proposer un schéma RELAX NG pour la structure.
3. Écrire un fichier d'exemple valide pour le schéma (avec au minimum 4 employés et une structure non triviale).
4. La structure hiérarchique de l'entreprise induit des contraintes qu'on ne peut pas traduire par l'intermédiaire du schéma (ou de la DTD). En d'autres termes, on peut écrire un fichier XML valide qui n'a pas de sens pratique. Donner un exemple d'un tel fichier.
5. Proposer un nouveau schéma RELAX NG n'utilisant pas de référence croisée et permettant de représenter une structure d'entreprise arborescente (i.e., chaque employé est le subordonné d'un seul autre employé, excepté le PDG). Donner un exemple de structure plausible qu'on ne peut pas représenter avec ce nouveau schéma mais qui ne pose pas de problème avec l'ancien.

Exercice 3.2

On souhaite représenter des expressions mathématiques simples dans un format XML proche de celui de MathML (un standard du W3C). Les principes du format proposé sont les suivants :

– on représente une valeur numérique comme le contenu d'un élément `cn`. Par exemple `<cn>15.2</cn>`
– une opération est représentée par un élément `apply`. En général l'élément contient trois sous-éléments. Le premier sous-élément est vide et représente l'opération à réaliser. Les deux autres éléments sont les opérandes de l'opération. Par exemple, $2+3$ s'écrit : `<apply><plus/><cn>2</cn><cn>3</cn></apply>`. La figure 1 représente l'arbre correspondant.
– on propose les opérateurs suivants :
  – `plus` : addition

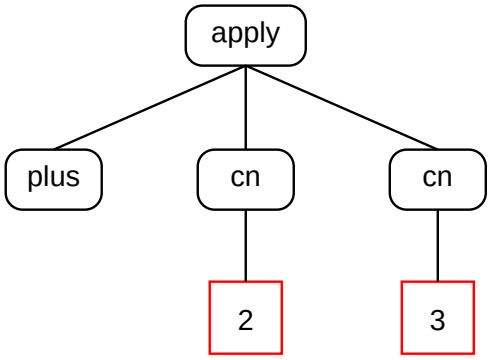

FIG. 1 – Arbre XML de l'expression $2 + 3$ représentée en MathML

- `times` : multiplication
- `divide` : division
- `minus` : soustraction
- `quotient` : division entière
- `rem` : reste de la division euclidienne
- `power` : exponentiation

Tous les opérateurs sont binaires sauf la soustraction qui peut aussi être unaire et représenter alors le changement de signe.

- un élément `apply` peut contenir un ou deux éléments `apply` comme opérande(s). Cette inclusion joue le rôle de parenthèses. Par exemple, pour représenter $(2 + 3) * 4$, on écrit :

equation1.xml

```
<apply>
  <times/>
    <apply>
      <plus/>
      <cn>2</cn>
      <cn>3</cn>
    </apply>
    <cn>4</cn>
</apply>
```

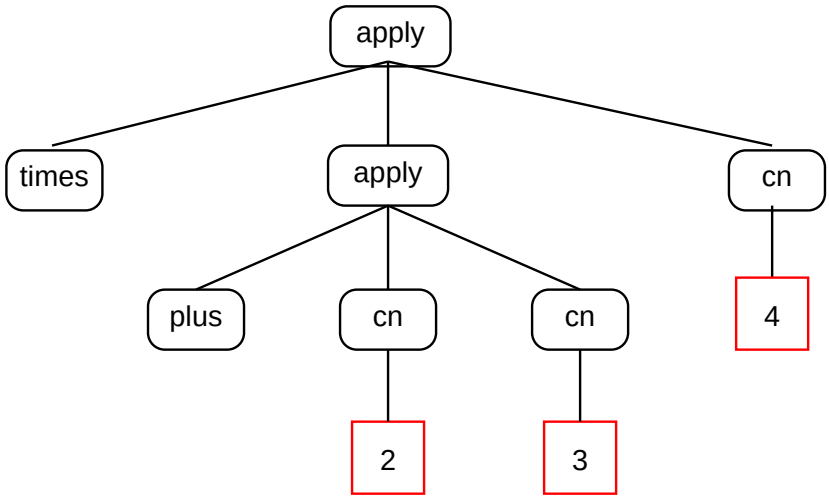

FIG. 2 – Arbre XML de l'expression $(2 + 3) * 4$ représentée en MathML

La figure 2 représente la structure logique correspondant à cette deuxièmte expression. Malgré la relative complexité de l'arbre obtenu, celui-ci reste assez proche de l'arbre qu'on peut déduire naturellement de l'expression mathématique, représenté par la figure 3.

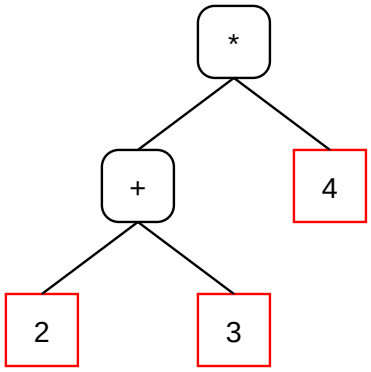

FIG. 3 – Arbre mathématique de l'expression $(2 + 3) * 4$

Questions :

1. Écrire un schéma RELAX NG pour le dialecte XML proposé.
2. Modifier le schéma obtenu en utilisant les types des schémas W3C pour introduire une contrainte supplémentaire sur le contenu des éléments.

Exercice 3.3

Le validateur RELAX NG Jing permet au concepteur d'un schéma d'ajouter des contraintes sur le contenu d'un document grâce à des classes de validation écrites en Java. Pour ce faire, le concepteur doit produire un fichier `jar` contenant :

- les fichiers `class` qui implantent les interfaces suivantes :
  - `DatatypeLibraryFactory` : assure la création d'une bibliothèque de types (à partir d'un URI)
  - `DatatypeLibrary` : représente la bibliothèque de types
  - `DatatypeBuilder` : assure la création d'un type (en tenant compte des paramètres associés)
  - `Datatype` : représente un type. C'est l'interface la plus importante qui est utilisée pour valider la conformité contenu du élément ou d'un attribut.
- un dossier `META-INF` contenant un dossier `services` contenant lui-même un fichier `org.relaxng.datatype.DatatypeLibraryFactory` dont la première ligne contient le nom de la classe qui implante `DatatypeLibraryFactory`.

Voici un exemple de programmation d'un nouveau type. On commence par implanter l'usine qui se contente de créer une `DatatypeLibrary` si l'URI qu'on lui propose correspond bien à l'URI associée à la bibliothèque programmée :

iup/app/IupDLF

```
1  package iup.app;
2  import org.relaxng.datatype.DatatypeLibraryFactory;
3  import org.relaxng.datatype.DatatypeLibrary;
4  public class IupDLF implements DatatypeLibraryFactory {
5    public DatatypeLibrary createDatatypeLibrary(String namespaceURI) {
6      // on reconnaît un ou plusieurs URI
7      if(namespaceURI.equals("http://apiacoa.org/teaching/xml/RelaxDataTypes")) {
8        return new IupDL();
9      }
10     return null;
11   }
12 }
```

La bibliothèque est implantée de façon basique car on ne gère qu'un seul type qui de surcroît n'utilise pas de paramètres :

iup/app/IupDL

```
1  package iup.app;
2  import org.relaxng.datatype.*;
3  import org.relaxng.datatype.helpers.ParameterlessDatatypeBuilder;
4  public class IupDL implements DatatypeLibrary {
5    public DatatypeBuilder createDatatypeBuilder(String baseTypeLocalName)
6      throws DatatypeException {
7      // pour l'instant, on fabrique des types qui n'utilisent pas de
8      // paramètres
9      return new ParameterlessDatatypeBuilder(createDatatype(baseTypeLocalName));
10   }
11   public Datatype createDatatype(String baseTypeLocalName)
12     throws DatatypeException {
13     // on ne gère qu'un seul type dans cette bibliothèque d'exemple
14     if(baseTypeLocalName.equals("telephone")) {
15       return new TypeTelephone();
16     } else {
17       throw new DatatypeException();
18     }
19   }
20 }
```

Le seul type proposé correspond à un modèle de contenu très strict : un numéro de téléphone à dix chiffres représenté sous forme de 5 blocs de deux chiffres séparés par un espace :

iup/app/TypeTelephone

```
1  package iup.app;
2  import org.relaxng.datatype.*;
3  import org.relaxng.datatype.helpers.StreamingValidatorImpl;
4  public class TypeTelephone implements Datatype {
5    public boolean isValid(String literal, ValidationContext context) {
6      // la méthode principale
7      // 66 66 66 66 66
8      if(literal.length()!=14)
9        return false;
10     for(int i=0;i<5;i++) {
11       int j=3*i;
12       if(!Character.isDigit(literal.charAt(j))) return false;
13       if(!Character.isDigit(literal.charAt(j+1))) return false;
14       if(i>=4) break;
15       if(literal.charAt(j+2)!=' ') return false;
16     }
17     return true;
18   }
19   // méthodes assez techniques sans intérêt en première approche
20   public DatatypeStreamingValidator
21     createStreamingValidator(ValidationContext context) {
22     return new StreamingValidatorImpl(this, context);
23   }
24   public Object createValue(String literal, ValidationContext context) {
25     if (!isValid(literal, context))
26       return null;
27     return literal;
```

```
28    }
29    public boolean sameValue(Object obj1, Object obj2) {
30      return obj1.equals(obj2);
31    }
32    public int valueHashCode(Object obj) {
33      return obj.hashCode();
34    }
35    public int getIdType() {
36      return ID_TYPE_NULL;
37    }
38    public boolean isContextDependent() {
39      return false;
40    }
41    public void checkValid(String literal, ValidationContext context)
42      throws DatatypeException {
43      if (!isValid(literal, context))
44        throw new DatatypeException();
45    }
46  }
```

Le fichier `org.relaxng.datatype.DatatypeLibraryFactory` contient l'unique ligne `iup.app.IupDLF`. Voici maintenant un exemple d'utilisation du type proposé dans un schéma RELAX NG :

telephone.rng

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <element name="root"
3           xmlns="http://relaxng.org/ns/structure/1.0">
4    <oneOrMore>
5      <element name="tel"
6        datatypeLibrary="http://apiacoa.org/teaching/xml/RelaxDataTypes">
7        <data type="telephone"/>
8      </element>
9    </oneOrMore>
10 </element>
```

Voici un fichier non valide (à cause du deuxième numéro de téléphone) :

telephone.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <root>
3   <tel>01 02 03 04 05</tel>
4   <tel>01-02-03-04-05</tel>
5  </root>
```

Questions :
Pour chaque question, on testera le résultat sur au moins un couple schéma/fichier XML.

1. Modifier `TypeTelephone.java` afin de proposer une validation plus souple, c'est-à-dire autoriser un nombre arbitraire d'espaces entre les chiffres d'un numéro (ainsi qu'avant et après le numéro).
2. Ajouter à la bibliothèque de types donnée en exemple un type `TypeCodePostal` qui correspond aux codes postaux français (on ne demande pas bien entendu un type parfait, mais au moins une validation des départements).
3. Ajouter à la bibliothèque de types un type `TypeSalleDauphine` qui correspond aux numéros des salles à Dauphine. Pour mémoire, il y a 7 étages et 5 zones (A, B, C, D et P), avec seulement 6 étages pour toutes les zones sauf A. Comme dans la question précédente, on veut obtenir une base de validation, pas une validation parfaite.

4. Pour prendre en compte des paramètres pour les types, on passe par l'intermédiaire d'une implantation d'un `DatatypeBuilder`. Par exemple, on modifie l'exemple du téléphone pour autoriser un nombre quelconque de blocs de 2 chiffres. On commence par modifier `TypeTelephone` afin de prendre en compte un nombre de blocs (on ne donne ici que le début de la classe, le reste n'étant pas modifié) :

iup/app/TypeTelephone

```
1  package iup.app;
2  import org.relaxng.datatype.*;
3  import org.relaxng.datatype.helpers.StreamingValidatorImpl;
4  public class TypeTelephone implements Datatype {
5    private int nbBlocs;
6    public TypeTelephone(int nb) {
7      nbBlocs=nb;
8    }
9    public boolean isValid(String literal, ValidationContext context) {
10     if(literal.length()!=3*nbBlocs-1)
11       return false;
12     for(int i=0;i<nbBlocs;i++) {
13       int j=3*i;
14       if(!Character.isDigit(literal.charAt(j))) return false;
15       if(!Character.isDigit(literal.charAt(j+1))) return false;
16       if(i>=nbBlocs-1) break;
17       if(literal.charAt(j+2)!=' ') return false;
18     }
19     return true;
20   }
21   // suite identique
```

Ensuite, on construit un `DatatypeBuilder` :

iup/app/TypeTelephoneBuilder

```
1  package iup.app;
2  import org.relaxng.datatype.*;
3  public class TypeTelephoneBuilder implements DatatypeBuilder {
4    private int nb=5;
5    public void addParameter(String name,String strValue,
6                             ValidationContext context)
7      throws DatatypeException {
8      // name -> nom du paramètre
9      // strValue -> valeur du paramètre
10     if(name.equals("nbBlocs")) {
11       try {
12         nb=Integer.parseInt(strValue);
13         if (nb<=0) throw new DatatypeException();
14       } catch (NumberFormatException e) {
15         throw new DatatypeException();
16       }
17     } else {
18       throw new DatatypeException();
19     }
20   }
21   public Datatype createDatatype() throws DatatypeException {
22     return new TypeTelephone(nb);
23   }
24 }
```

On termine en modifiant la `DatatypeLibrary` :

iup/app/IupDL

```
package iup.app;
import org.relaxng.datatype.*;
import org.relaxng.datatype.helpers.ParameterlessDatatypeBuilder;
public class IupDL implements DatatypeLibrary {
  public DatatypeBuilder createDatatypeBuilder(String baseTypeLocalName)
    throws DatatypeException {
    if(baseTypeLocalName.equals("telephone")) {
      return new TypeTelephoneBuilder();
    } else {
      throw new DatatypeException();
    }
  }
  public Datatype createDatatype(String baseTypeLocalName)
    throws DatatypeException {
    return createDatatypeBuilder(baseTypeLocalName).createDatatype();
  }
}
```

Voici maintenant un exemple d'utilisation du type proposé dans un schéma RELAX NG :

telephone2.rng

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element name="root"
         xmlns="http://relaxng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="tel"
      datatypeLibrary="http://apiacoa.org/teaching/xml/RelaxDataTypes">
      <data type="telephone">
        <param name="nbBlocs">4</param>
      </data>
    </element>
  </oneOrMore>
</element>
```

Voici un fichier non valide (à cause du deuxième numéro de téléphone) :

telephone2.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
<tel>01 02 03 04</tel>
<tel>01 02 03 04 05</tel>
</root>
```

En utilisant le mécanisme proposé, modifier le type `TypeTelephone` pour permettre le choix du nombre de chiffres ainsi que celui des séparateurs autorisés entre les chiffres (en plus de l'espace). La validation se fera de façon souple, c'est-à-dire comme à la question 1.

5. Ajouter à la bibliothèque de types un type `TypeEntier` qui correspond à un contenu entier. On proposera deux paramètres facultatifs permettant de réduire l'intervalle autorisé pour les valeurs entières (un paramètre `min` et un paramètre `max`).
6. Ajouter à la bibliothèque de types un type `ListeFichier` qui correspond à un texte choisi dans une liste de textes contenue dans un fichier. Plus précisément, le type est associé à un paramètre obligatoire (`file`) qui donne le nom d'un fichier. Le validateur doit lire le fichier (qui contient un texte autorisé par ligne) et vérifier que le contenu de l'élément ou de l'attribut est bien dans la liste des textes autorisés. Il est très vivement conseillé de faire la lecture une seule fois au moment de la création du type (des stratégies plus évoluées de cache sont d'ailleurs possibles).

4 Schémas W3C

Exercice 4.1

On souhaite représenter un emploi du temps sous forme de fichier XML. Pour chaque plage de temps occupée, on souhaite conserver les informations suivantes :
- description
- date et horaires
- lieu
- catégorie

De plus, on souhaite gérer les répétitions sans redondance dans le fichier, c'est-à-dire en associant à un évènement une liste de dates au lieu d'une seule date. Pour compresser une telle liste, on souhaite pouvoir indiquer qu'un évènement se répète un certain nombre de fois, d'une date à une autre, etc., en tenant compte aussi d'exceptions à une règle simple (par exemple tous les jeudis de telle date à telle autre, sauf le jeudi tant).

Questions :

1. Donner un schéma W3C pour le format proposé.
2. Donner un fichier XML qui illustre toutes les possibilités du dialecte proposé.