Chapitre V

Formalisation - XMLSchema : Schema et Instance

Un XMLSchema (xsd) permet de définir:

- un nouveau vocabulaire (les mots clef du langage)
- un ensemble de règles (grammaire) précisant comment utiliser les mots du vocabulaire (i.e. comment ils s'articulent entre eux)

Un schéma XML (XMLSchema) sert donc de modèle pour la création de documents XML (ce que l'on nomme "instance").

0.a Exemple

On considère par exemple le document suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<position>
    <module>32.904237</module>
    <angle>73.620290</angle>
</position>
```

Pour faciliter le traitement automatisé de ce type de document, on peut souhaiter imposer quelques *règles* sur les données de tous les documents de ce type. Ce document suggère des données sous forme de coordonnées polaires[1]. Un `module` devrait donc être:

- un nombre
- réel
- positif

L'`angle` quant à lui doit être compris soit entre -180 et 180 degrés, soit entre $-\pi$ et π (radians). Selon ce ce que l'on veut exprimer par ces données (le modèle de données), on pourra éventuellement ajouter d'autres règles/restrictions.

Pour garantir l'efficacité d'un algorithme sur ces données, on va vouloir valider de manière automatique la conformité du document XML (des données) par rapport aux règles (modèle sous forme XMLShema) comme illustré Figure V.1

[1] http://fr.wikipedia.org/wiki/Coordonn%C3%A9es_polaires

0.b Validité d'un document XML

Un document XML est dit **valide** par rapport à un modèle de données décrit dans un document XMLSchema (XSD) si et seulement si:

- il est bien formé
- il est conforme au modèle défini par le XSD, à savoir:
  - l'élément racine porte le bon nom
  - l'ensemble des éléments est structuré, les éléments organisés dans le bon ordre
  - les données sont du type attendu

0.c Modèle et Instance

Un **XMLSchema** est aussi appelé

- un **modèle** de données
- un vocabulaire/langage XML
- un **espace de noms XML**[2]
- un document XSD (pour *Xml Schema Description*). xsd est également l'extension des noms de fichiers XMLSchema

Un **document XML** valide *par rapport à un XMLSchema* est aussi appelé une **instance** du schéma/vocabulaire XSD.

Remarque 1:

XMLSchema est lui-même un langage XML. Un Schéma XML est donc également un document XML valide par rapport au XMLShema des XMLSchemas...[3]. On peut ainsi résumer la validation de documents XML par rapport à un schéma grâce à la figure V.2

Remarque 2:

En pratique, la validation se fait par l'intermédiaire d'un programme tiers (en Java, C++, ...) qui (i) analyse le XMLSchema, (ii) en constuit un modèle interne, (iii) analyse le document XML dont on doit vérifier la validité et (iv) vérifie la correspondance entre le schéma XML et le document XML.

[2] nous reviendrons sur cette notion au chapitre suivant
[3] nous reviendrons sur cette notion en TP...

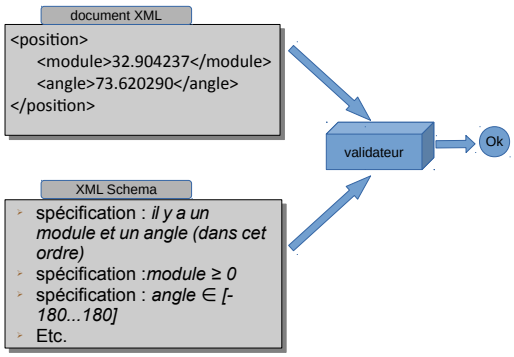

Figure V.1: Exemple de Validateur XMLShema.

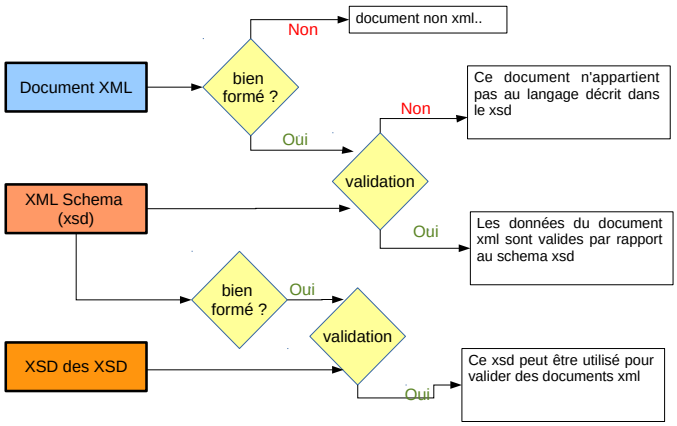

Figure V.2: Étapes de validation d'un document et d'un schéma XML.

Remarque 3:

On peut faire un vrai parallèle entre le Schéma et l'Instance XML, et la Classe et l'Instance d'objet en programmation objet. Ce parallèle peut vous aider à vous rappeler le rôle de chaque partie du code.

Par exemple, si l'on considère le schéma (faites attention aux commentaires) suivant :

```
<?xml version="1.0"?>
<schema version="1.0"
           xmlns="http://www.w3.org/2001/XMLSchema"
           xmlns:dst="http://www.timc/fr/nicolas.glade/distance"
           targetNamespace="http://www.timc/fr/nicolas.glade/distance"
           elementFormDefault="qualified">

    <!-- Déclaration de deux éléments racine différents nommés "distance"
         et "dist". Ces éléments seront les racines de 2 instances
         différentes évidemment ! -->
    <!-- Ceci est équivalent en C#/Java à la déclaration d'une variable
         dans le code. Notez que tp, le namespace, correspond au
         package/namespace en Java/C# :
         /* déclaration d'une variable distance et d'une variable dist de type
            Distance la classe Distance appartenant au package/namespace dst */
         dst.Distance distance;
         dst.Distance dist ;
    -->
    <element name="distance" type="dst:Distance"/>
    <element name="dist" type="dst:Distance"/>

    <!-- Définition du type Distance -->
    <!-- Equivalent à la définition d'une classe C#
         namespace dst;
         enum Unite {km, m, mm}
         class Distance {
                public float Value { get; set; }
                Unite unite;
         }
    -->
    <complexType name="Distance">
        <simpleContent>
            <extension base="nonNegativeInteger">
                <attribute name="unite" type="dst:Unite" use="required"/>
            </extension>
```

```
        </simpleContent>
    </complexType>

    <!-- Définition du type restreint Unite -->
    <!-- Equivalent à une enumération en Java/C# -->
    <simpleType name="Unite">
        <restriction base="string">
            <enumeration value="km"/>
            <enumeration value="m"/>
            <enumeration value="mm"/>
        </restriction>
    </simpleType>

</schema>
```

et une instance possible :

```
<?xml version="1.0" encoding="UTF-8"?>
<dst:dist
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:dst='http://www.timc/fr/nicolas.glade/distance'
    xsi:schemaLocation='http://www.timc/fr/nicolas.glade/distance Distance.xsd'
    unite="km">380</dst:dist>
```

... ou bien celle-ci :

```
<?xml version="1.0" encoding="UTF-8"?>
<d:distance
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:d='http://www.timc/fr/nicolas.glade/distance'
    xsi:schemaLocation='http://www.timc/fr/nicolas.glade/distance Distance.xsd'
    unite="mm">0.31</d:distance>
```

Faisons le parallèle avec un programme en C# en commençant par la définition du type `Distance` :

```
namespace dst;

public class Distance {
    public enum Unite { mm, m, km}
    public float _value { set; get; }
    public Unite _unite { init; get; }

    public Distance(float value, Unite unite) {
        _unite = unite;
        _value = value;
    }
}
```

ou en style plus C# en faisant de `Distance` un contrusteur primaire :

```
namespace dst;

public class Distance(float value, Distance.Unite unite) {
       public enum Unite { mm, m, km}
       public float _value { set; get; } = value;
       public Unite _unite { init; get; } = unite;
   }
```

et le programme appelant dans lequel on déclare une variable de type distance, puis ont instancie :

```
using dst;

internal class Program {
    public static void Main(string[] args) {
        Distance dist; // déclaration
```

```
6           dist = new Distance(380, Distance.Unite.km); // instanciation +
                affectation
7       }
8   }
```

Le comparatif est immédiat :

- La définition de classe C# correspond à la définition de type complexe (complexType) en XMLSchema
- Le namespace C# nommé `dst` correspond au namespace (xmlns) XMLSchema nommé `http://www.timc/fr/nicola` et préfixé `dst`
- La déclaration de la variable C# `distance` de type `dst.Distance` correspond à la déclaration d'un élément racine dans le Schema XML `<element name="distance" type="dst:Distance"/>`
- L'instanciation en C# (`new Distance(380, Distance.Unite.km)`) correspond au contenu du fichier `Distance.xml` avec ses valeurs de distance et d'unité particulières.
- L'affectation en C# (`dist = ...`) correspond à l'existance du fichier `Distance.xml` dans le système.