Chapitre IV

Formalisation - But des Schemas XML

1 Objectifs

L'informatique est la science du traitement et de la manipulation de données. Dans le chapitre précédent, nous avons vu comment était formé un document XML. Dans ce cours, nous allons voir comment organiser des données dans un document XML particulier, le Schema XML.

Organiser des données XML sur la base de la description textuelle d'une situation (par exemple l'énumération des employés d'une société avec leurs relations hiérarchiques, ou encore la modélisation d'eaux minérales de compositions différentes, de conteneurs différents ...) peut être ambigu. En effet, il peut y avoir plusieurs solutions possibles qui modélisent plus ou moins bien la question.

Il est alors difficile de prévoir un traitement automatique sans savoir exactement quelles données seront utilisées, ni quelles seront leurs valeurs ou opérations possibles. C'est pourquoi nous allons utiliser des langages de modélisation.

Dans ce cours, nous utiliserons les diagrammes UML et langage XMLSchema pour la modélisation des données.

Grâce au diagramme UML (*Unified Modeling Language*), nous allons pouvoir expliquer un système d'information sans ambiguïté. Le langage XMLSchema, permet également de décrire les types de données (de quelles données ils sont composés, et quels sont les types primitifs de ces données) et leur organisation (comment ces types sont composés entre eux), sous forme textuelle. Ce format permettra par la suite de valider un document XML par rapport à son schéma.

L'intérêt de cette démarche est double :

- pour le concepteur (ou l'équipe de conception), cela permet de réfléchir *a priori* (ce qu'il faut TOUJOURS FAIRE) à la façon dont on va organiser nos données (et par extension, nos programmes ; savoir formaliser un problème orienté données est un préalable à savoir formaliser un projet de programmation plus vaste). On spécifie **à l'avance** le problème ou les données du problème = on dit comment on pense le problème / les données (comment elles seront typées et agencées). Cette réflexion préliminaire est indispendable pour mener à bien un projet sur le long terme et éviter des déconvenues à cause de mauvaises planifications. Ainsi une fois que les choses sont fixées par un tel schéma, une telle formalisation, il faut s'y conformer. S'il s'avérait que la modélisation devenait bloquante pour la suite du projet car finalement pas adaptée, cela necessiterait de se rassoir autour d'une table et réflechir à comment faire évoluer tout ce qui dépend de cette struturation ; dans ce cas le "bricolage *a posteriori*" n'est pas permis.

- pour le concepteur (ou l'équipe de conception) et pour l'utilisateur, ceci sert à containdre *a posteriori* à l'échelle d'un projet les fichiers de données et les classes sérialisées dans un programme exploitant ces données. Cela permet une standardisation, par exemple pour permettre à plusieurs équipes de travailler sur un même type de données. Par exemple, à la CNAM (la Caisse Nationale d'Assurance Maladie, un éditeur de logiciels pour les caisses primaires (CPAM)), un individu est

modélisé de la même manière dans les différents services (arrêts de travail, naissance d'un enfant, maladies de longue durée ...), son modèle étant forcé par un seul et même fichier : le schéma.

2 Spécifier

Ce qui vient d'être dit dans la section précédente et qui a déjà été dit dans un chapitre précédent, implique que **lorsqu'on programme, on ne doit pas se contenter d'aligner des instructions de telle manière que "ça marche"** mais on doit **d'abord, planifier, typer, organiser ...**.

La meilleure façon de procéder, et vous y serez confrontés dans votre projet de fin d'année (et le cas échéant, votre projet échouera !), c'est **d'abord de discuter entre vous** comment chacun voit l'organisation d'un ensemble de données, d'un programme ..., et très vite ...**de faire des schémas ... sur papier**. Oui, car on pense plus vite sur papier que sur ordinateur et parce que sur un papier on peu gribouiller, gommer, ... [*disclaimer* : votre prof datant d'un autre siècle, il utilise du papier, mais bien entendu, ça marche avec une tablette et un stylet]. Ca, c'est la première étape.

La seconde, c'est d'être plus précis en spécifiant exactement ce qu'on a, ce qu'on veut ... pour chaque élément d'une donnée ou d'un programme, pour chaque étape d'un programme. Etre précis, c'est raconter une petite histoire : "ce type contient ceci et cela et leur type est ainsi ou comme ça ..." ou bien pour un programme (ou une transformation XSLT par exemple) "cette fonction/transformation utilise tel argument pour faire ça, puis fait ceci, puis fait cela, puis renvoie cette valeur dont le type est tsointsoin". Et cette histoire, on l'écrit ... **DANS LES COMMENTAIRES !!!!!!!!!!!!!!** Oui, ça sert à ça !

Et maintenant mon conseil : lorsque vous écrivez une classe, un type XMLSchema, une transformation XSLT, **(1)** d'abord vous réflechissez sur papier, **(2)** ensuite vous écrivez un long commentaire racontant ce que contient ou fait votre bout de code, **(3)** enfin et seulement après vous écrivez effectivement votre bout de code !

Exemple 1. la modélisation des bagages

1) D'abord, on fait un schéma UML (à la main ou avec un outil numérique) de notre modèle de liste de bagages pour un vol.

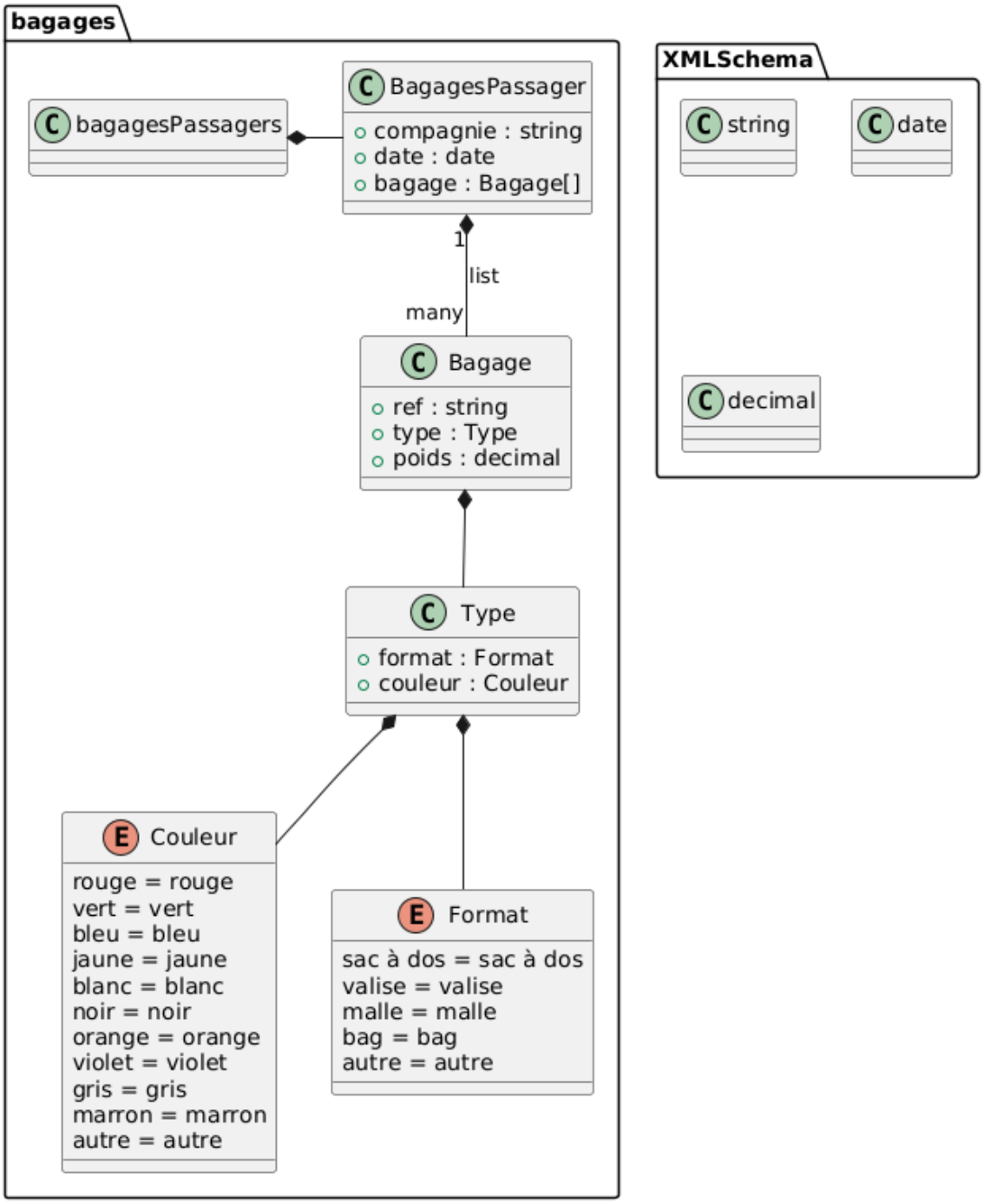

Figure IV.1: Diagramme de classes modélisant les données sur les bagages.

2) Ensuite on crée le document XMLSchema et on écrit les commentaires.

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

        <!-- ................. RACINE ................... -->
```

```
9         <!-- On déclare un élément nommé bagagesPassager,
10                 de type BagagesPassager, qui sera l'élement racine de notre
                       instance XML -->
11
12
13        <!-- ................. TYPES ..................... -->
14        <!-- Le type BagagesPassagers contient :
15                - une compagnie de type string
16                - une date au format date standard
17                - un ensemble illimité d'élements bagage (de type Bagage). Il
                      peut y avoir 0 bagages -->
18
19
20    <!-- Le type Bagage contient :
21                - une référence de type Ref (un type simple restreint à un
                      pattern)
22                - un type de bagage de type Type (un type complexe contenant un
                      format de bagages et une couleur)
23                - un poids de type décimal -->
24
25
26    <!-- Le type Type contient :
27                - un format de bagages de type Format (un type simple enuméré)
28                - une couleur de type Couleur (un type simple enuméré) -->
29
30
31    <!-- Le type Format est une restriction de string proposant une énumération
32                de différents formats de bagages, par ex une valise ou un sac à
                       dos -->
33
34
35    <!-- Le type Couleur est une restriction de string proposant un
36                certain nombre de couleurs plausibles pour des bagages -->
37
38
39    <!-- Le type Ref est une restriction de string obeissant à une regex telle
          que :
40                - la référence commence par 2 lettres en majuscule
41                - elle est suivie d'un nombre indéterminé de chiffres -->
42
43
44 </schema>
```

3) Le listing ci-dessous montre l'étape suivante, un exemple de Schema XML construit à partir des spécifications faites dans les commentaires. Ce schema modélise les données qui sont instanciées dans le document `bagages.xml`

```
1  <?xml version="1.0"?>
2  <schema version="1.0"
3             xmlns="http://www.w3.org/2001/XMLSchema"
4             xmlns:bg="http://www.timc.fr/nicolas.glade/bagages"
5             targetNamespace="http://www.timc.fr/nicolas.glade/bagages"
6             elementFormDefault="qualified">
7
8         <!-- ................. RACINE ..................... -->
9         <!-- On déclare un élément nommé bagagesPassager,
10                 de type BagagesPassager, qui sera l'élement racine de notre
                       instance XML -->
11    <element name="bagagesPassagers" type="bg:BagagesPassager"/>
12
13
```

```
14          <!-- ................. TYPES ..................... -->
15          <!-- Le type BagagesPassagers contient :
16                  - une compagnie de type string
17                  - une date au format date standard
18                  - un ensemble illimité d'élements bagage (de type Bagage). Il
                        peut y avoir 0 bagages -->
19      <complexType name="BagagesPassager">
20          <sequence>
21              <element name="compagnie" type="string"/>
22              <element name="date" type="date"/>
23              <element name="bagage" type="bg:Bagage" minOccurs="0" maxOccurs="
                    unbounded"/>
24          </sequence>
25      </complexType>
26
27      <!-- Le type Bagage contient :
28                  - une référence de type Ref (un type simple restreint à un
                        pattern)
29                  - un type de bagage de type Type (un type complexe contenant un
                        format de bagages et une couleur)
30                  - un poids de type décimal -->
31      <complexType name="Bagage">
32          <sequence>
33              <element name="ref" type="bg:Ref"/>
34              <element name="type" type="bg:Type"/>
35              <element name="poids" type="decimal"/>
36          </sequence>
37      </complexType>
38
39      <!-- Le type Type contient :
40                  - un format de bagages de type Format (un type simple enuméré)
41                  - une couleur de type Couleur (un type simple enuméré) -->
42      <complexType name="Type">
43          <attribute name="format" type="bg:Format" use="required"/>
44          <attribute name="couleur" type="bg:Couleur"/>
45      </complexType>
46
47      <!-- Le type Format est une restriction de string proposant une énumération
48                  de différents formats de bagages, par ex une valise ou un sac à
                        dos -->
49      <simpleType name="Format">
50          <restriction base="string">
51              <enumeration value="sac à dos"/>
52              <enumeration value="valise"/>
53              <enumeration value="malle"/>
54              <enumeration value="bag"/>
55              <enumeration value="autre"/>
56          </restriction>
57      </simpleType>
58
59      <!-- Le type Couleur est une restriction de string proposant un
60                  certain nombre de couleurs plausibles pour des bagages -->
61      <simpleType name="Couleur">
62          <restriction base="string">
63              <enumeration value="rouge"/>
64              <enumeration value="vert"/>
65              <enumeration value="bleu"/>
66              <enumeration value="jaune"/>
67              <enumeration value="blanc"/>
68              <enumeration value="noir"/>
69              <enumeration value="orange"/>
70              <enumeration value="violet"/>
71              <enumeration value="gris"/>
```

```
            <enumeration value="marron"/>
            <enumeration value="autre"/>
        </restriction>
    </simpleType>

    <!-- Le type Ref est une restriction de string obeissant à une regex telle
        que :
                - la référence commence par 2 lettres en majuscule
                - elle est suivie d'un nombre indéterminé de chiffres -->
    <simpleType name="Ref">
        <restriction base="string">
            <pattern value="[A-Z]{2}[0-9]*"/>
        </restriction>
    </simpleType>

</schema>
```

Exemple 2. modélisation d'une transformation des données de bagages en page html

1) ce qu'on veut (une page html qu'on prototype à la main ; ici c'est le résultat qui est montré).

Bagages du vol Air France du 2012-01-06

Référence	Description	Poids enregistré
AF48717	sac à dos de couleur noir	7.3
AF677793	sac à dos de couleur rouge	9.8
AF67840	valise de couleur noir	22.5

Figure IV.2: Page HTML que l'on souhaite voir afficher et dont on va modéliser les transformations.

2) Les commentaires pour modéliser la feuille de transformation des données de bagages en page html.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    xmlns:bg="http://www.timc.fr/nicolas.glade/bagages">

    <xsl:output method="html"/>

    <!-- Le template principal
        - contient tous les éléments du fichier html (entête, body ...)
        - définit le style de la table (bordures blanches simples et cellules
            bleu-vert) dans le header
        - affiche un titre "Bagages du vol **compagnie** du **date**
        - crée une table avec en première ligne (tr) l'entête contenant 3
            cellules (th) (la "Référence", la "Description", le "Poids enregistr
            é)
        - applique les templates sur le champ sélectionné bg:bagage en triant
            les bagages par numéro de référence croissant
    -->


```

```
    <!-- Ce template s'applique sur les éléments bg:bagages
       Il crée une ligne de table html (tr) dans laquelle il crée 3 cellules (td)
        - la référence
        - la description au format "**format de bagage** de couleur **la couleur**"
        - le poids du bagage
    -->


</xsl:stylesheet>
```

3) La feuille de transformation des données de bagages en page html.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    xmlns:bg="http://www.timc.fr/nicolas.glade/bagages">

    <xsl:output method="html"/>

    <!-- Le template principal
        - contient tous les éléments du fichier html (entête, body ...)
        - définit le style de la table (bordures blanches simples et cellules bleu-vert) dans le header
        - affiche un titre "Bagages du vol **compagnie** du **date**
        - crée une table avec en première ligne (tr) l'entête contenant 3 cellules (th) (la "Référence", la "Description", le "Poids enregistré)
        - applique les templates sur le champ sélectionné bg:bagage en triant les bagages par numéro de référence croissant
    -->
    <xsl:template match="/">
        <html>
            <head>
                <title>bagages.xsl</title>
                <style>
                table, th, td {
                    border: 1px solid white;
                    border-collapse: collapse;
                    }
                th, td {
                    background-color: #96D4D4;
                    }
                </style>
            </head>
            <body>
                <h2>Bagages du vol <xsl:value-of select="//bg:compagnie/text()" /> du <xsl:value-of select="//bg:date/text()"/></h2>
                <table>
                    <tr>
                        <th>Référence</th>
                        <th>Description</th>
                        <th>Poids enregistré</th>
                    </tr>
                    <xsl:apply-templates select="//bg:bagage">
                        <xsl:sort select="bg:ref"/>
                    </xsl:apply-templates>
                </table>
            </body>
        </html>
```

```
43      </xsl:template>
44
45      <!-- Ce template s'applique sur les éléments bg:bagages
46          Il crée une ligne de table html (tr) dans laquelle il crée 3 cellules (
                td)
47           - la référence
48           - la description au format "**format de bagage** de couleur **la
                couleur**"
49           - le poids du bagage
50      -->
51      <xsl:template match="bg:bagage">
52          <tr>
53              <td><xsl:value-of select="bg:ref/text()"/></td>
54              <td><xsl:value-of select="bg:type/@format"/> de couleur <xsl:value-
                    of select="bg:type/@couleur"/></td>
55              <td><xsl:value-of select="bg:poids/text()"/></td>
56          </tr>
57      </xsl:template>
58
59  </xsl:stylesheet>
```