Chapitre II

Formalisation - Introduction

1 Formaliser / Modéliser

1.a Stocker

Le stockage de données est évidemment un problème crucial sur Internet ou localement dans nos ordinateurs.

Supposons l'énoncé suivant : "On veut stocker dans un fichier la liste des bagages d'un passager d'un vol Air France. Le passager a 2 bagages qu'il enregistre le 6 janvier 2012. Le premier bagage est un sac à dos rouge référencé AF677793 et le second une valise noire référencée AF67840. Ces bagages pèsent respectivement 9.8 et 22.5 kg".

Comment stocker ces données ? Comment les modéliser ? Il s'agit de 2 problèmes différents.

Il y a autant de façons de stocker les données que de programmeurs. Chaque programmeur peut décider de son propre format de données, le programme qui lira ou écrira ces données étant intrinsèguement lié à la façon dont elles sont arrangées dans le document de sauvegarde.

Par exemple on pourrait avoir le document montré figure II.1. On voit là que ce document n'a pas une organisation très claire. Un autre exemple est le format CSV montré figure II.2. Cette fois, la première ligne contient le nom de champs et les autres les valeurs... pourquoi pas. Le format CSV est après tout très utilisé lorsque les données sont assez massives et destinées à faire l'objet de traitements statistiques par exemple.

```
Air France
2012-01-06
bagage AF677793
sac à dos ; rouge ; 9.8
bagage AF67840
valise ; noir ; 22.5
```

Figure II.1: Exemple de document texte stockant des données.

```
Compagnie   Date         RefBagage   Type        Couleur Poids
Air France  2012-01-06   AF677793    sac à dos   rouge   9.8
Air France  2012-01-06   AF67840     valise      noir    22.5
```

Figure II.2: Autre exemple de document texte stockant des données au format CSV (coma separated text).

1.b Modéliser

Mais ces représentations ne reposent pas sur des modèles orientés objets des données. On voit bien dans ces exemple que l'ensemble représente un grand objet, l'ensemble des bagages d'un passager d'Air France dont le vol est le 6 janvier 2012, et la liste de ses bagages (chaque bagage étant un objet)... En programmation Orienté Objet, par exemple en Java ou C#, vous n'auriez aucun mal à imaginer leur **modélisation** sous forme de classes, par exemple :

```
1  package bagages;
2
3  import java.time.LocalDate;
4  import java.util.List;
5
6  public class BagagesPassager {
7
8       enum Format {sac_a_dos,valise,malle,bag,autre}
9       enum Couleur {rouge, vert, bleu, jaune, blanc, noir, orange, violet, gris,
            marron, autre}
10
11         public class Type{
12             public Format format;
13             public Couleur couleur;
14             }
15
16         public class Bagage {
17             public String ref;
18             public Type type;
19             public float poids;
20             }
21
22     public String compagnie;
23     public LocalDate date;
24     public List<Bagage> bagage;
25 }
```

Figure II.3: Représentation objet de la liste des bagages d'un passager en Java (objet ne comprortant que des données ; pas de méthodes).

Cette fois, l'espace des données est intelligement découpé et organisé : on sait qui contient quoi et quels sont les types des valeurs qui seront stockées dans les attributs. Et plus que ça encore : on a cette fois modélisé les données *indépendamment* des valeurs associées. Le programmeur est parti du principe que les bagages, les types, les couleurs, particuliers pouvaient être représentés par des modèles <=> formalisés.

On doit donc voir plus loin que la valeur et la généraliser en types ... et ensuite organiser ces types. C'est cela modéliser. C'est ce que nous ferons dans cette UE.

1.c UML

Mais avant d'implémenter un modèle dans un langage dédié comme XML Schema, il est utile de représenter ces types (le modèle) et valeurs (instances) sous une forme graphique. C'est à cela que sert UML (Unified Modeling Language). UML est une norme graphique permettant de représenter de nombreuses choses en informatique : diagrammes de classes, arbres d'instances, diagrammes fonctionnels ...

Nous n'allons pas faire ici un cours d'UML mais au fil de ces cours nous montrerons comment représenter Schemas XML, diagrammes de classes, instances XML ... sous la forme de diagrammes UML.

Au delà de la représentation à l'aide d'un papier et d'un crayon, ou d'un tableau et de feutres, outils principaux d'un bon programmeur, notez qu'il existe plusieurs outils utiles pour générer de tels diagrammes. PlantUML en fait partie. Pour s'en servir, il suffit de soumetttre en ligne (Web Server PlantUML) votre code PlantUML. Vous trouverez de la documentation ici et ici.

... et notez aussi qu'il existe des plugins intéressants comme par exemple celui qui génère des diagrammes de classes à partir de schemas XML. C'est d'ailleurs avec ce plugin que j'ai généré le code plantuml ci-dessous correspondant au Schema XML donné figure II.8, à l'aide de la commande suivante : `xsdata bagages/bagages.xsd -output plantuml -package bagages` (NB: en réalité, le plugin n'a pas généré les packages et quelques relations que j'ai ajoutés).

```
@startuml

  package XMLSchema {
    class string
    class date
    class decimal
  }

  package bagages {

    class Bagage {
        +ref : string
        +type : Type
        +poids : decimal
    }

    class BagagesPassager {
        +compagnie : string
        +date : date
        +bagage : Bagage[]
    }

    enum Couleur {
        rouge = rouge
        vert = vert
        bleu = bleu
        jaune = jaune
        blanc = blanc
        noir = noir
        orange = orange
        violet = violet
        gris = gris
        marron = marron
        autre = autre
    }

    enum Format {
        sac à dos = sac à dos
        valise = valise
        malle = malle
        bag = bag
        autre = autre
    }

    class Type {
        +format : Format
        +couleur : Couleur
    }

    class bagagesPassagers {}

    bagagesPassagers *- BagagesPassager
    Type *-- Format
    Type *-- Couleur
    BagagesPassager "1" *-- "many" Bagage : list
    Bagage *-- Type

```

```
58
59 }
60 @enduml
```

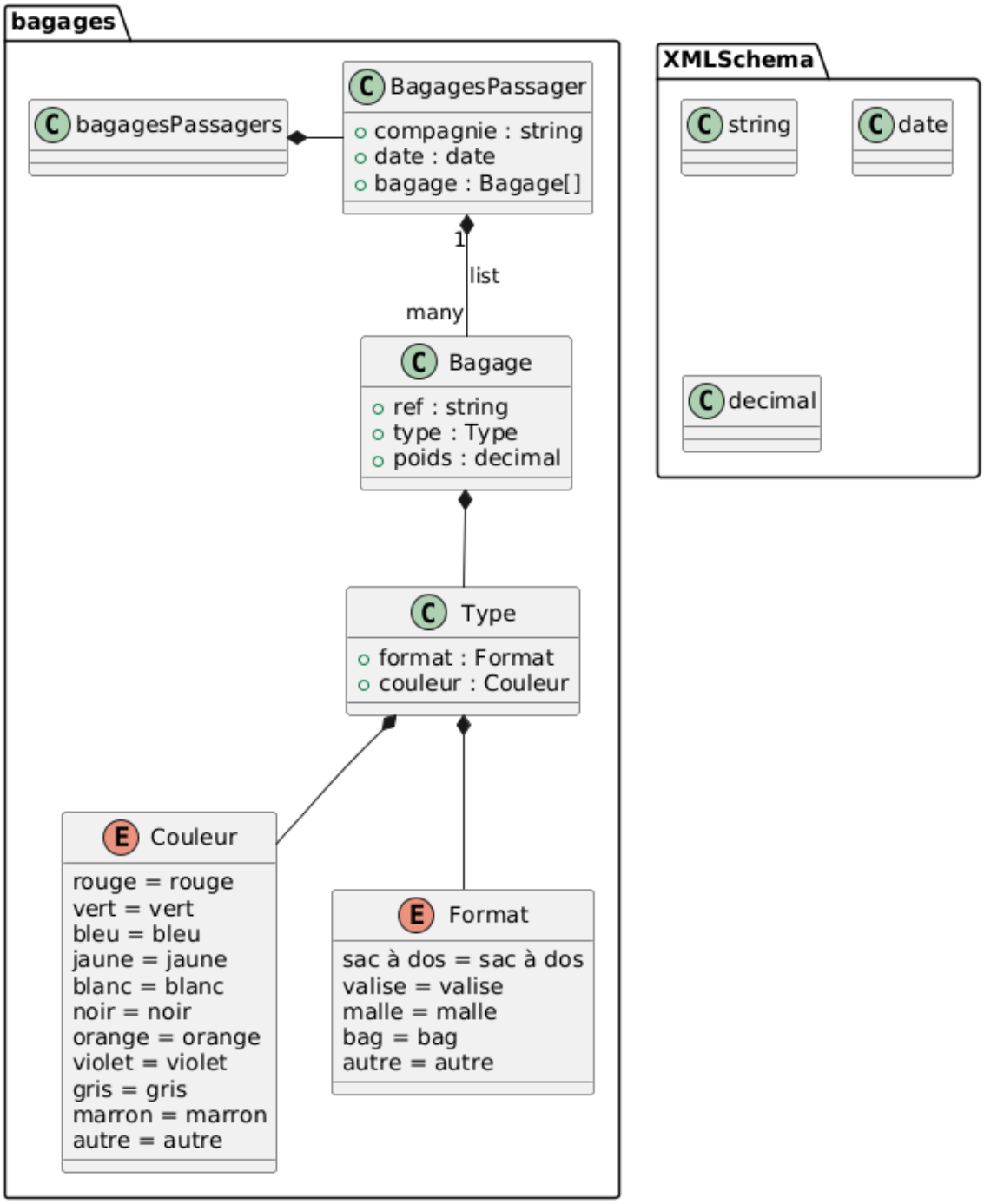


Figure II.4: Diagramme de classes généré par le code PlantUML préalablement généré et correspondant au Schema XML donné figure II.8.

2 Les données et les langages de formalisation

XML est un langage à `balises` comme l'est `HTML`. Certains d'entre vous connaissent HTML ; d'autres vont découvrir ce langage, notamment dans cette UE. La ressemblance entre ce que l'on écrit en XML et en HTML risque de vous induire en erreur : HTML est un langage dédié à la présentation des données ; XML est un langage de modélisation et de stockage des données. Les balises utilisées dans HTML (ex: `div`, `p`, `b`, `i`, `br` ...) ont un sens pour les navigateurs ; elles n'ont `STRICTEMENT AUCUN SENS` (sauf celui qu'on leur donnera) en XML. Réciproquement, les balises marquant le début et la fin des éléments en XML n'ont aucun sens pour les navigateurs ; elles n'ont du sens que pour le modélisateur et ceux qui exploitent les données via un langage appelant.

Ci-dessous nous allons présenter HTML/XHTML, un langage de présentation des données, pour comprendre facilement la notion de balise et surtout parce que vous en aurez besoin lorsque nous ferons du XSLT. Nous présenterons aussi rapidement JSON, puis XML, les deux étant des langages de modélisation et stockage des données.

2.a Les langages html et xhtml - des langages de présentation

HTML est l'acronyme d' *Hypertext Markup*[1] *Language*. C'est un langage pour écrire des pages web. Plus généralement, c'est un langage pour présenter des documents ... **MAIS** il n'est pas fait pour décrire ou structurer des données.

Une *page web* est un document lisible par une application: le navigateur (par exemple Mozilla Firefox, Internet Explorer, Safari, Opera, etc.). C'est un fichier qui est stocké sur un disque dur soit local, soit distant et nous est rendu disponible par transfert de données grâce à l'Internet via le protocole HTTP[2]. En pratique, pour accéder à une page web via le protocole HTTP, on tape dans le navigateur l'URL du fichier (cf Figure II.5).

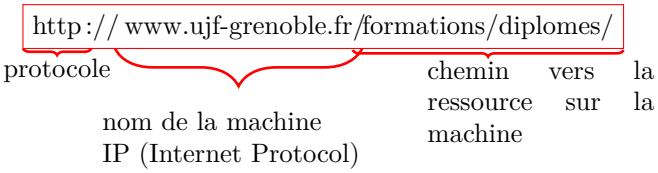


Figure II.5: URL: Uniform Ressource Locator.

Un document **HTML** est un document hyper-texte (images, vidéos, boutons, ...) qui est organisé grâce à des balises, marqueurs spécifiques du langage indiquant comment le texte doit être formaté, disponible, etc.

Un document **XHTML** (eXtended HTML) est un document HTML conforme au format XML (c'est un document XML dont le vocabulaire utilisé est du HTML et dont la structure respecte à la fois celle du XML et du HTML). Cela reste un langage de présentation des données et non de structuration.

2.b Structure d'un document XHTML

Prologue, racine, espace de nom

Le langage XHTML est un document XML avant tout ! il a donc tous les marqueurs d'un document XML (voir chapitre suivant : XML). En particulier, il dispose d'un prologue, d'une racine unique nommée `html`, et précise l'usage du namespace `http://www.w3.org/1999/xhtml` indiquant que le vocabulaire utilisé est celui de XHTML :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>TODO supply a title</title>
```

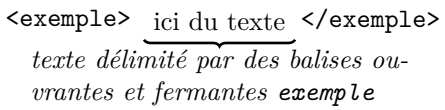
[1] Markup = Balise en anglais
[2] Hyper Text Transfert Protocol

```
        <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

Les balises

Le langage XHTML est organisé grâce à des balises.
`<nom>` est une balise ouvrante (on ouvre un bloc d'organisation). `nom` est alors le nom de la balise.
`</nom>` est une balise fermante. Une balise ouvrante et une balise fermante délimitent un bloc.
`<nom/>` est une balise ouvrante/fermante (dites encore auto-fermante).

Le XHTML est un ensemble de texte et de marqueurs délimitant des blocs:




```
<exemple>
    <plus> compliqué </plus>
</exemple>
```

On a, dans ce second exemple, une structure arborescente que l'on peut représenter comme suit:

```
+ exemple
|
 - + plus
   |
    - compliqué
```

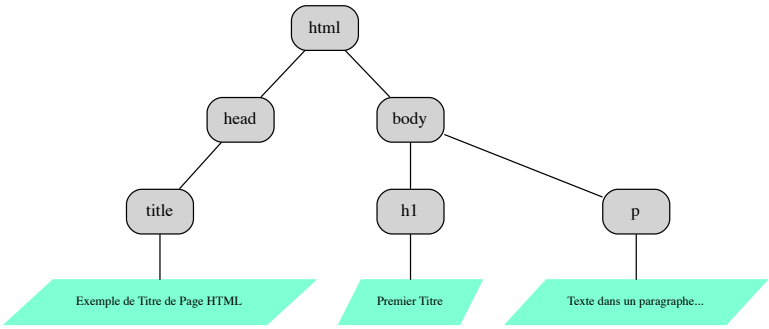
Structure HTML

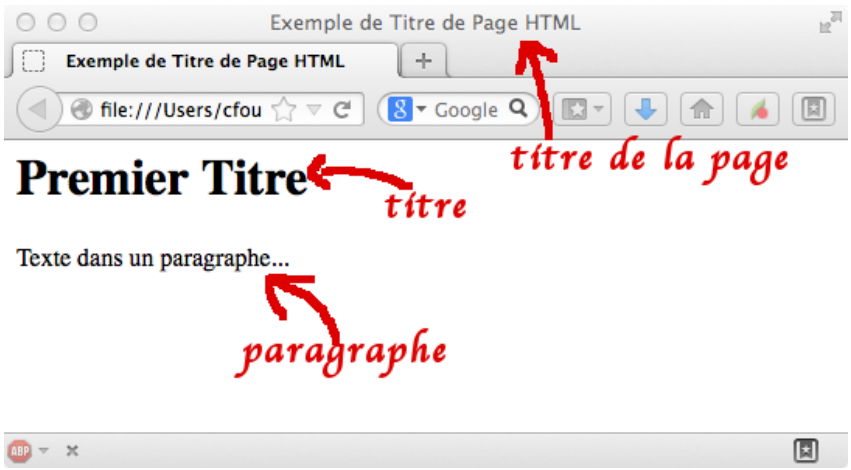
HTML définit un ensemble de balises (mots) ainsi qu'une organisation (sémantique). HTML utilise la balise `html` pour englober toutes les autres balises.

```
1 <html>
2     <head>
3         <title>Exemple de Titre de Page HTML</title>
4     </head>
5     <body>
6         <h1>Premier Titre</h1>
7         <p>Texte dans un paragraphe...</p>
8     </body>
9 </html>
```

Cet exemple de page HTML peut aussi être représentée comme suit:

```
+ html
|
 - + head
|  |
|   - + title
|  |  |
|  |   - Exemple de Titre de Page HTML
|  |
|   - + etc.
|
 - + body
   |
    - + h1
   |  |
   |   - Premier Titre
   |
    - + p
  |
   - Texte dans un paragraphe...
```





Les principales balises html

`html` Balise *racine* d'un document html. Les autres balises sont incluses dans cette balise

`head` Entête d'un document html. Contient des informations sur le document qui ne seront pas forcément affichées dans le navigateur.

`body` Indique le corps du document contenant ce qui doit être affiché sur la page web. Les balises suivantes sont inclues dans la balise `body`

1. Les balises de structure

   `h1` Titre de niveau 1 (gros)

   `h2` Titre de niveau 2 (plus petit)

   `p` Paragraphe

   `br` Retour à la ligne

   `hr` Ligne horizontale

2. Mise en forme de texte

   `i` Italique (exemple: `<i>mot</i>`)

   `b` Gras (exemple: `<b>texte en gras</b>`)

3. Les listes

`ol` Listes numérotées (*orderded list*)

```
<ol>
  <li> Point A </li>
  <li> Point B </li>
  <li> C  </li>
</ol>
```

1. Point A
2. Point B
3. C

`ul` Liste à points (*unordered list*)

```
<ul>
  <li> Point A </li>
  <li> Point B </li>
  <li> C  </li>
</ul>
```

- Point A
- Point B
- C

4. Les tableaux

   `table` Début d'un tableau

   `tr` Ligne (*table raw*)

   `td` Cellule d'une ligne (*table data*)

```
<table>
  <tr>
    <td>A</td>
    <td>B</td>
  </tr>
  <tr>
    <td>C</td>
    <td>D</td>
  </tr>
</table>
```

A	B
C	D

5. Les hyperliens

   `a` élément d'hyperlien avec un attribut `href` (*anchor*)

```
<a href="http://google.com"> cliquer ici </a>
```

   Il y a un espace entre le nom de l'élément `a` et le nom de l'attribut `href`. Si l'on clique sur le texte `cliquer ici`, on arrive sur la cible du lien référencé par l'attribut `href` (ici la page d'accueil de Google).

6. Les images

   `img` élément d'insertion du image avec un attribut `src` (*image*).

```
<img src="img2141.jpg"/>
```

   On peut remarquer que la balise `img` est une balise ouvrante/fermante. L'attribut `src` a pour valeur le nom du fichier image à insérer. Ce nom peut contenir un chemin relatif sur le disque ou bien une URL.

2.c La limite d'HTML

HTML, surtout doublé de feuilles de styles CSS, est un formidable langage pour modéliser des présentations de documents. Ses spécifications (structure, nom des balises ...) sont précises ... mais ne peuvent pas être ni changées, ni étendues, ni utilisées pour modéliser des données ! Et c'est bien ça la limite de ce langage !
HTML ne peut pas servir à représenter des données, à les typer, à les stocker. Pour faire cela, il faut utiliser un langage dédié comme JSON ou XML.

3 JSON - un langage de modélisation et stockage des données

JSON signifie JavaScript Object Notation. C'est un standard utilisé pour représenter des données structurées. Il est particulièrement compatible avec Javascript dont il reprend la syntaxe des objets. Il peut cependant être utilisé séparément de Javascript. Ce n'est pas particulièrement un langage "à balises", mais dans une certaine mesure, l'enchassement des différents objets et leur nommage explicite s'y apparente.

Nous n'allons pas faire un cours de JSON ici, d'autant que ce ne serait pas bien compliqué ; vous trouverez assez facilement des tutos en ligne. L'idée est de vous montrer les différences de documents XML et JSON, et vous montrer aussi les différences de typage entre XMLSchema et JSO?-Schema. JSON est plus compact que XML en écriture mais finalement assez peu. En réalité, vous découvrirez assez tôt l'intérêt majeur que représentent les technologies XML, bien plus vastes que ce que permet l'usage de JSON, ces technologies XML existant depuis plus longtemps et plus stables que les technologies JSON.

3.a Structure d'un document JSON

Un objet JSON sont délimités par des accolades ouvrantes et fermantes, les valeurs numériques sont notées sans guillemets, les valeurs Booléennes également (true ou false) et les textes entre guillemets, les tableaux sont délimités par des crochets. Les champs contenant les valeurs sont nommés et les valeurs leurs sont associées par le symbole **:**. Dans un même objet, les champs sont séparés par des virgules. Par exemple :

```
{
  "compagnie" : "Air France",
  "date": "2012-01-06",
  "bagages" :
  [
    {
      "ref" : "AF677793",
      "type" :
      {
        "format" : "sac à dos",
        "couleur" : "rouge"
      },
      "poids" : 9.8
    },
    {
      "ref" : "AF67840",
      "type" :
      {
        "format" : "valise",
        "couleur" : "noir"
      },
      "poids" : 22.5
    }
  ]
}
```

Figure II.6: Exemple de document JSON

Voila, vous savez écrire du JSON ! Mais vous verrez qu'écrire du XML n'est pas plus compliqué.

3.b JSON Schema

Les types stockés ici (dans le document JSON) peuvent être modélisés. Certains aspects de la modélisation ne sont cependant pas simples à implémenter en JSON Schema, en particulier la notion d'héritage.

Voila un exemple très simplifié par rapport au schéma XML correspondant (voir plus loin) de ce qu'on pourrait écrire en JSON Schema. On voit que ce n'est pas simple à écrire et dur à lire. On pourrait l'améliorer en faisant plusieurs schémas et surtout, il manque dans ce schéma la contrainte (expression

régulière) sur la référence bagage, ainsi que les types énumérés pour la couleur du bagage et son type. On peut le faire bien entendu. Je vous laisserai faire votre choix etre les schémas XML et JSON ...

```
{
  "$id": "http://www.timc.fr/nicolas.glade/bagages",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "compagnie": {
      "description": "le nom de la compagnie",
      "type": "string"
    },
    "date": {
      "description": "la date du vol",
      "type": "date"
    },
    "bagages": {
      "description": "une liste de bagages",
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "ref": {
            "description": "la référence d'un bagage",
            "type": "string"
          },
          "type": {
            "type": "object",
            "properties": {
              "format": {
                "description": "le type de bagage",
                "type": "string"
              },
              "couleur": {
                "description": "la couleur d'un bagage",
                "type": "string"
              }
            }
          },
          "poids": {
            "description": "le poids d'un bagage",
            "type": "number"
          }
        }
      },
      "minItems": 1,
      "uniqueItems": true
    }
  },
  "required": [
    "compagnie",
    "date",
    "bagages"
  ]
}
```

4 XML - un langage de modélisation et stockage des données

Un ensemble de technologies associées à un langage permettent de lever les ambiguïtés et de bien distinguer le fond et la forme d'un document, contrairement à HTML ou même XHTML, et de modéliser des données : les technologies XML.

XML signifie E**X**tensible **M**arkup **L**anguage, c'est à dire *langage de balises qui peut être étendu*. Nous verrons par la suite que l'on peut définir nous-même le nom des balises (contrairement au HTML où, par exemple, les balises délimitant un paragraphe sont déjà définies et ne peuvent être que `<p>....</p>`). C'est en cela qu'XML est extensible. Cependant, comme son vocabulaire n'est pas défini *a priori*, il ne s'agit pas vraiment d'un langage informatique, mais d'un *méta-langage* (c'est-à-dire un langage qui permet de définir des langages, ou *dialectes*).

Il s'agit en fait simplement d'un ensemble de règles à respecter lorsque l'on veut décrire des informations structurées.

4.a Exemple

On considère l'exemple de la figure II.7 :

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Bagagerie -->
<bagagesPassagers
    xmlns="http://www.timc.fr/nicolas.glade/bagages"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.timc.fr/nicolas.glade/bagages bagages.xsd">
    <compagnie>Air France</compagnie>
    <date>2012-01-06</date>
    <bagage>
        <ref>AF677793</ref>
        <type format="sac à dos" couleur="rouge"/>
        <poids>9.8</poids>
    </bagage>
    <bagage>
        <ref>AF67840</ref>
        <type format="valise" couleur="noir"/>
        <poids>22.5</poids>
    </bagage>
</bagagesPassagers>
```

Figure II.7: Exemple de document XML (*instance xml*).

On peut remarquer plusieurs choses sur ce document, par exemple:

- le nom des balises est en français, ce qui est peu commun pour un langage informatique. Il s'agit en effet d'un vocabulaire *ad hoc*, qui a été défini spécialement par la personne qui a écrit ce document pour une application donnée.
- le balises sont *indentées*. Lorsqu'une balise commence à l'intérieur d'un autre élément, elle est décalée vers la droite pour une meilleur lisibilité. Indenter correctement son document XML n'est pas obligatoire, mais cela fait partie des bonnes pratiques de programmation.
- le nom des éléments est explicite: même si l'on ne connaît pas ce langage, on peut comprendre aisément le type d'information contenu dans les balises. C'est un document lisible par tous.
- les balises ne décrivent que la sémantique (ce que sont les choses, par exemple *AF67840* est une référence de bagage d'un passager), il n'y a aucune information de présentation des données (en gras, italique, retour à la ligne, etc.).
- il se réfère à un autre document (bagages.xsd) et fait appel à plusieurs vocabulaires, ou namespaces, (ici le vocabulaire des bagages défini dans le fichier `bagages.xsd` et nommé `http://www.timc.fr/nicolas.glade/bagages`, ainsi que le vocabulaire pour l'instance de schema, nommé `http://www.w3.org/2001/XMLSchema-instance`). Nous verrons plus loin dans le cours comment sont nommés ces vocabulaires et comment on s'y réfère.

On a ici un exemple de données aéroportuaires décrites dans un langage XML spécifique. En informatique, **un exemple est aussi appelé une *instance***. Cette instance est fondée sur un nouveau langage qui permet d'ordonner et de typer les informations jugées nécessaires pour une application donnée.

A cette instance de document est associée un **Schema XML** (défini dans le fichier `bagages.xsd`, avec `xsd` pour XML Schema Descriptor), autrement dit un type qui, à l'instar des classes en langage objet et de leurs attributs, modélise sous la forme de types (types complexes ou simples) les objets de données qui sont instanciés dans le document XML. Voici ce schéma (ci-dessous). Etudiez le en regardant comment il est construit, et mettez le en regard de l'instance XML ci-dessus.

```xml
<?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">

    <element name="bagagesPassagers" type="bg:BagagesPassager"/>

    <complexType name="BagagesPassager">
        <sequence>
            <element name="compagnie" type="string"/>
            <element name="date" type="date"/>
            <element name="bagage" type="bg:Bagage" minOccurs="0" maxOccurs="
               unbounded"/>
        </sequence>
    </complexType>

    <complexType name="Bagage">
        <sequence>
            <element name="ref" type="bg:Ref"/>
            <element name="type" type="bg:Type"/>
            <element name="poids" type="decimal"/>
        </sequence>
    </complexType>

    <complexType name="Type">
        <attribute name="format" type="bg:Format" use="required"/>
        <attribute name="couleur" type="bg:Couleur"/>
    </complexType>

    <simpleType name="Format">
        <restriction base="string">
            <enumeration value="sac à dos"/>
            <enumeration value="valise"/>
            <enumeration value="malle"/>
            <enumeration value="bag"/>
            <enumeration value="autre"/>
        </restriction>
    </simpleType>

    <simpleType name="Couleur">
        <restriction base="string">
            <enumeration value="rouge"/>
            <enumeration value="vert"/>
            <enumeration value="bleu"/>
            <enumeration value="jaune"/>
            <enumeration value="blanc"/>
            <enumeration value="noir"/>
            <enumeration value="orange"/>
            <enumeration value="violet"/>
            <enumeration value="gris"/>
            <enumeration value="marron"/>
            <enumeration value="autre"/>
        </restriction>
    </simpleType>

    <simpleType name="Ref">
        <restriction base="string">
            <pattern value="[A-Z]{2}[0-9]*"/>
        </restriction>
    </simpleType>

</schema>
```

Figure II.8: Exemple de Schema XML. Ce schema modélise les données qui sont instanciées dans le document `bagages.xml`.