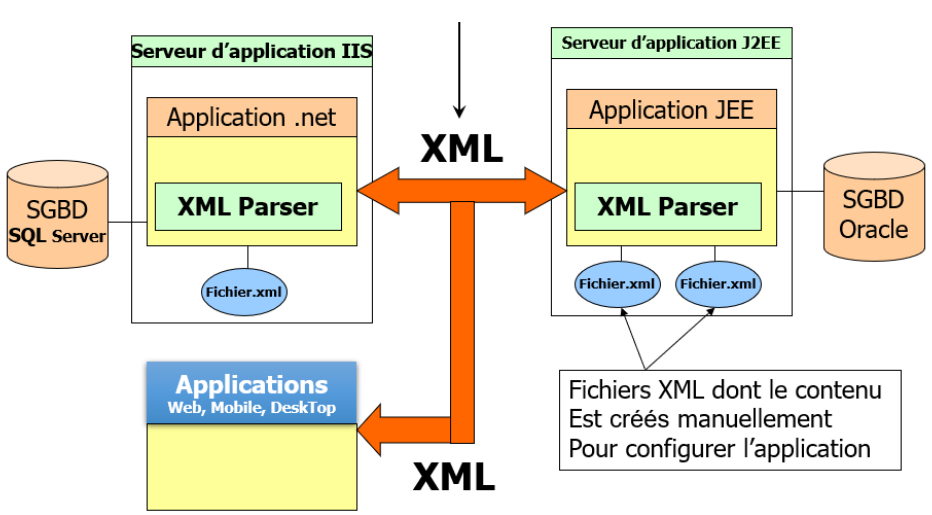
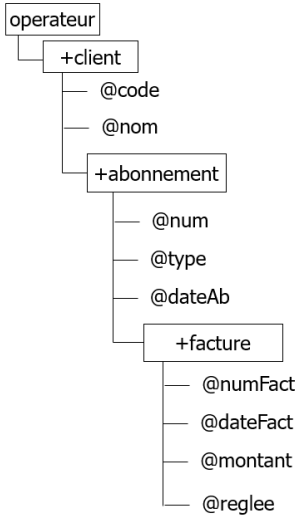
Bases de données semi-structurées et Web Services
Technologie XML : XML, DTD, XSL, XSD, Xpath, XQuery, SVG

Master Ingénierie Informatique, Big Data et Cloud Computing II-BDCC,
ENSET Mohammedia , Université Hassan II de Casablanca






```
ml version="1.0" encoding="UTF-8"?>
erateur>
 <client code="1" nom="hassouni">
     <abonnement num="123" type="GSM" dateAb="2006-1-11">
         <facture numFact="432" dateFact="2006-2-11" montant="350.44" reglee="oui"/>
         <facture numFact="5342" dateFact="2006-3-11" montant="450" reglee="oui"/>
         <facture numFact="5362" dateFact="2006-4-13" montant="800.54" reglee="non"/>
     </abonnement>
     <abonnement num="2345" type="FIXE" dateAb="2006-12-1">
         <facture numFact="5643" dateFact="2007-1-12" montant="299" reglee="oui"/>
         <facture numFact="6432" dateFact="2007-2-12" montant="555" reglee="non"/>
     </abonnement>
 </client>
 <client code="2" nom="abbassi">
     <abonnement num="7543" dateAb="2006-12-1" type="GSM">
         <facture numFact="7658" dateFact="2007-2-12" montant="350.44" reglee="oui"/>
         <facture numFact="7846" dateFact="2007-3-12" montant="770" reglee="non"/>
     </abonnement>
 </client>
perateur>
```

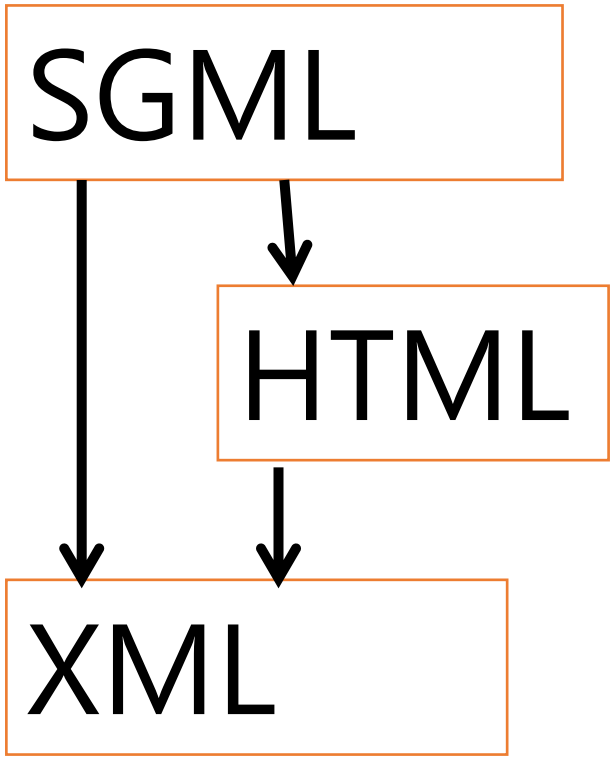

Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Origine de XML

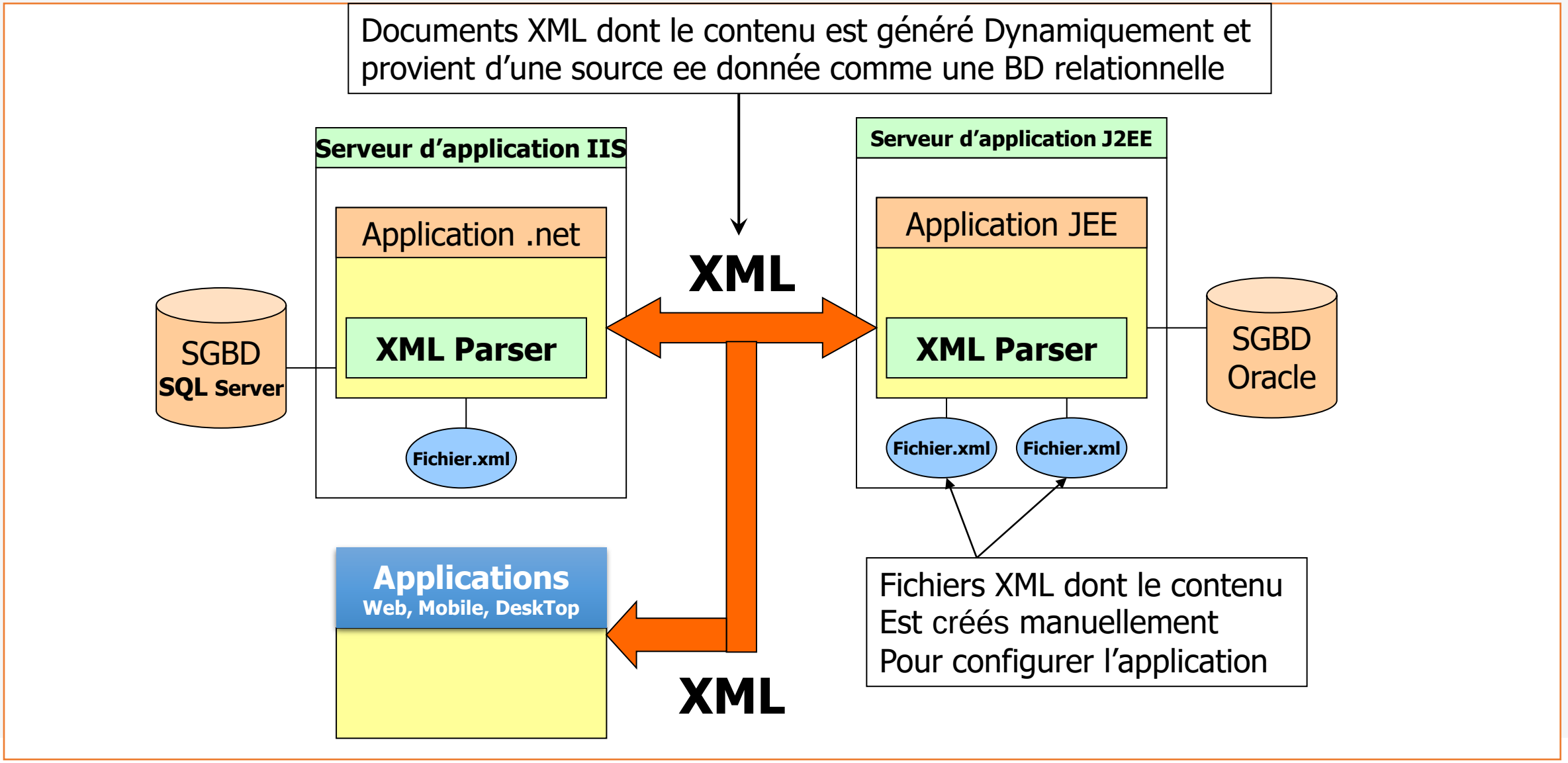
Standard Generalized Markup Language
(Sépare les données la structure des données et la mise en forme)

Hyper Text Markup Language
(Mélange les données et la mise en forme)

eXtensible Markup Language
(Sépare les données la structure des données et la mise en forme)

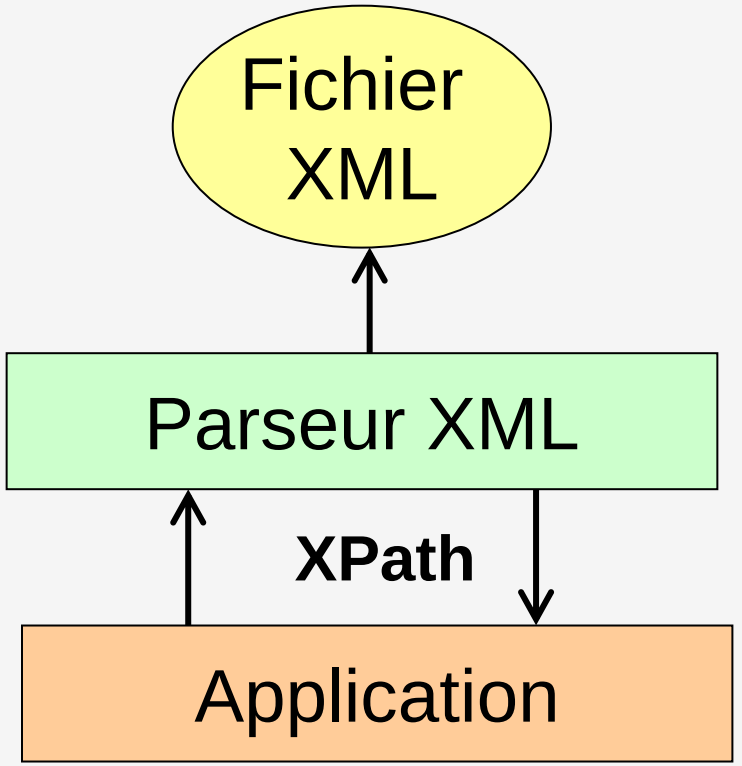


XML pour l'échange de données entre les Systèmes d'Information

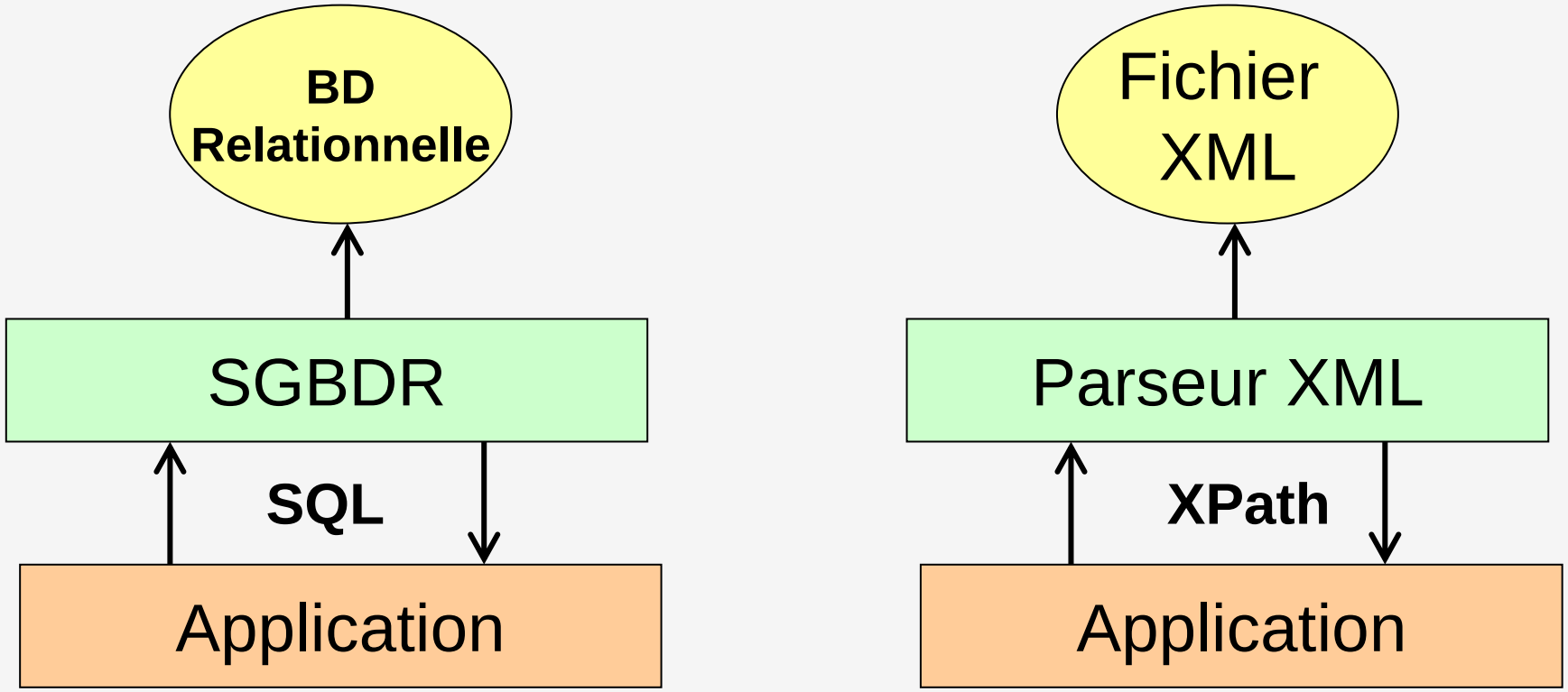


XML ?

- XML est un langage d'échange de données structurés entre applications de différents systèmes d'informations.
- Les données d'un fichier XML sont organisée d'une manière hiérarchique
- Les données d'un fichier XML peuvent provenir des bases de données relationnelles. (Documents XML Dynamiques)
- Les fichiers XML sont également utilisés en tant que fichiers de configuration d'une application. (Documents XML Statiques)
- Pour lire un fichier XML, une application doit utiliser un parseur XML.
- Un parseur XML est une API qui permet de parcourir un fichier XML en vue d'en extraire des données précises.

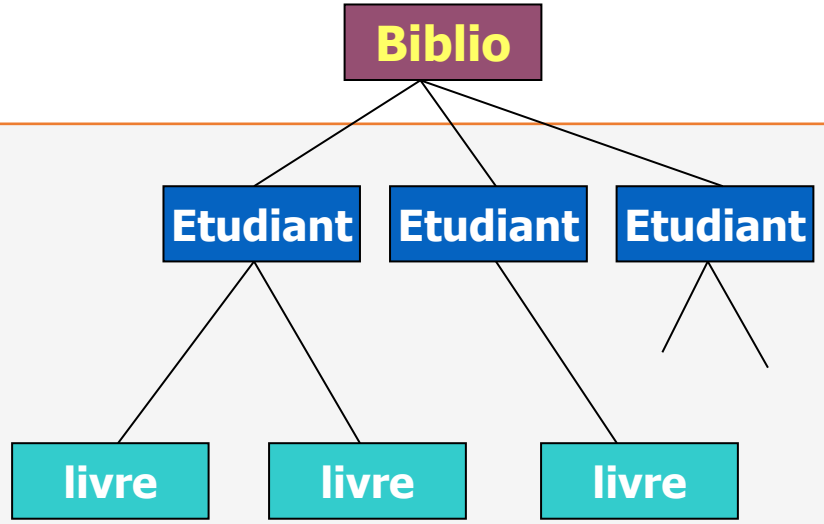


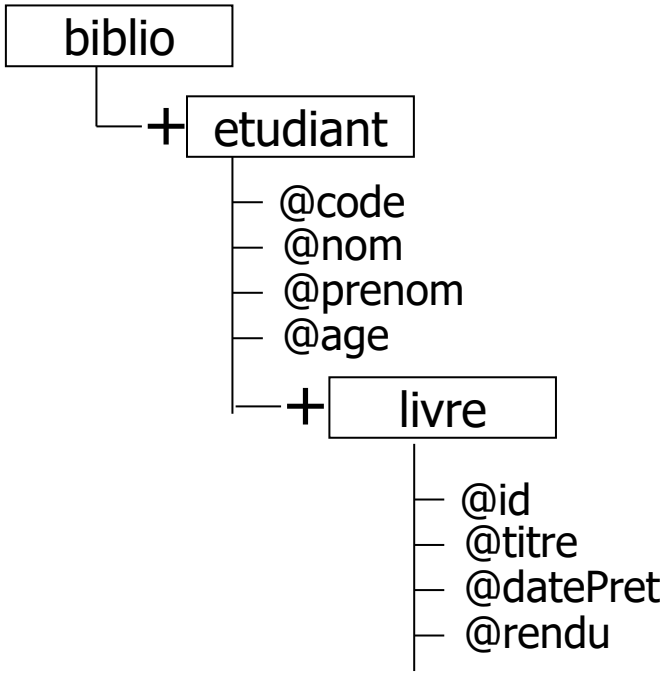
Correspondance entre XML et Bases de données relationnelles



Exemple de document XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<biblio>
   <etudiant code="1" nom="A" prenom="B" age="23">
      <livre id="534" titre="java" datePret="2006-11-12" rendu="oui"/>
      <livre id="634" titre="XML" datePret="2006-11-13" rendu="non"/>
   </etudiant>
   <etudiant code="2" nom="C" prenom="D" age="22">
      <livre id="33" titre="E-Commerce" datePret="2006-1-12" rendu="non"/>
   </etudiant>
</biblio>
```

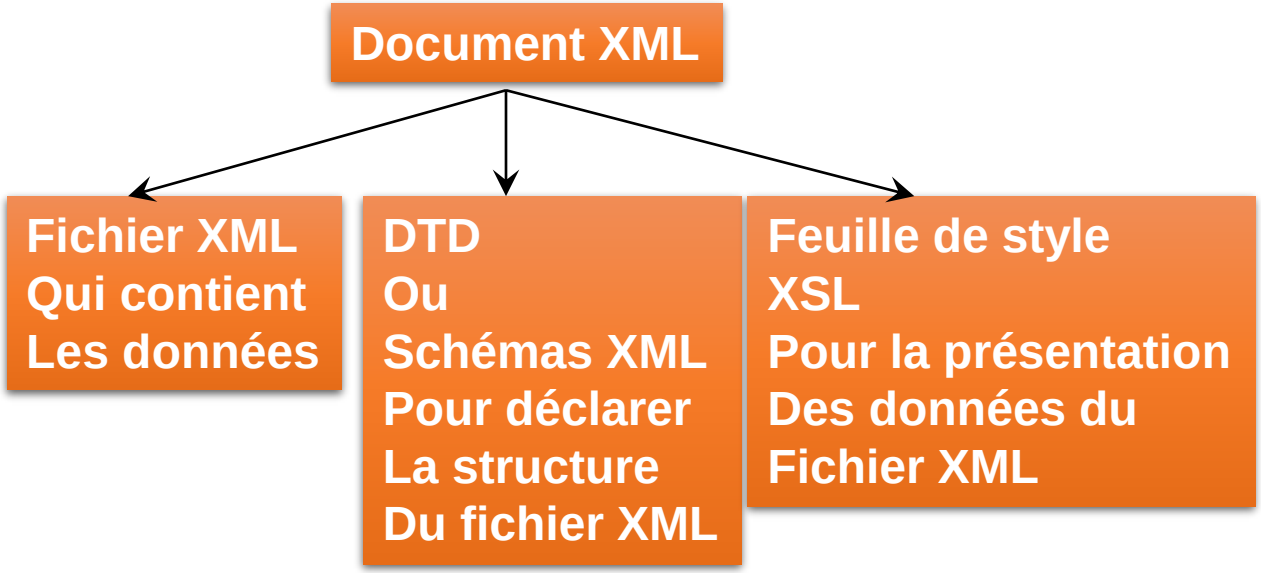




Représentation graphique de l'arbre XML

Structure d'un document XML

Un document XML se compose de 3 fichiers :



- Le fichier XML stocke les données du document sous forme d'un arbre
- DTD (Data Type Definition) ou Schémas XML définit la structure du fichier XML
- La feuille de style définit la mise en forme des données de la feuille xml

Syntaxe d'un document XML

Un document XML est composé de deux parties :

- **Le prologue**, lui même composés de plusieurs parties
  - Une **déclaration XML**, qui permet de définir :
    - la version de XML utilisée,
    - le codage des caractères
    - la manière dont sont stockées les informations de balisage

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

  - Une **déclaration de type de document** (DTD) qui permet de définir la structure logique du document et sa validité.

```
<!DOCTYPE racine  SYSTEM "produit.dtd">
```

  - Une **transformation** définie par une feuille de style (CSS ou XSL) qui permet de définir la présentation.

```
<?xml-stylesheet type="text/xsl" href="produit.xsl"?>
```

- **L'instance** qui correspond au balisage du document proprement-dit (Données sous forme d'arbre).

Terminologies XML

Balise :

- C'est un mot clef choisi par le concepteur du document qui permet de définir un élément.
  - Exemple : **`<formation>`**

Elément :

- C'est un objet XML défini entre une balise de début et une balise de fin. La balise de fin porte le même nom que la balise de début, mais elle est précédée d 'un "slash".
  - **`<formation>`**
  - **`Contenu de l 'élément`**
  - **`</formation>`**
- Un élément peut contenir aussi d'autre éléments

Attribut :

- Une élément peut être qualifié par un ou plusieurs attributs. Ces attributs ont la forme clef="valeur".
  - **`<formation code="T03" type="qualifiante" >`**

Document XML bien formé

- Pour qu'un document soit bien formé, il doit obéir à 4 règles :

1. Un document XML ne doit posséder qu'une seule racine
2. Tous les éléments doivent être fermés
3. Les éléments contenus et contenant doivent être imbriqués.
4. La valeurs des attributs s'écrit entre guillemets

Document XML bien formé

Un document XML ne doit posséder qu'un seul élément racine qui contient tous les autres.

- Un document XML est un arbre.

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  TSIG Etudes
</formation>
<durée>
  1755 heures
</durée>
```

Document bien formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée>
    1755 heures
  </durée>
</formation>
```

Document XML bien formé

Tous les éléments doivent être fermés

- A chaque balise ouvrant doit correspondre une balise fermante.
- A défaut, si un élément n'a pas de contenu, il est possible d'agréger la balise fermante à la balise ouvrant en terminant celle-ci par un "slash".

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755">
</formation>
```

Document bien formé:

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
</formation>
```

Document XML bien formé

Les éléments contenus et contenant doivent être imbriqués.

- Tous les éléments fils doivent être contenus dans leur père.
- Si un document XML est un arbre, un élément est une branche.

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
  <module id="100">
  <sequence id="525">
  </module>
  </sequence>
</formation>
```

Document bien formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
  <module id="100">
    <sequence id="110">
    </sequence>
  </module>
</formation>
```

Document XML bien formé

La valeurs des attributs s'écrit entre guillemets.

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur=1755 />
  <module id=100>
  </module>
  <module id=110>
  </module>
</formation>
```

Document bien formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
  <module id="100">
  </module>
  <module id="110">
</module>
</formation>
```

Document XML valide

Un document XML valide est un document XML bien formé dont la structure respecte les déclarations définie dans :

- Une DTD (*Document Type Definition*)
- Un Schéma XML (XSD)

La DTD est une série de spécifications déterminant les règles structurelles des documents xml qui lui sont associés.

- La DTD spécifie :
  - Le nom des balises associées à tous les éléments,
  - Pour chaque balise, les attributs possibles et leur type,
  - Les relations contenant-contenu entre les éléments et leur cardinalité,
  - Les entités (raccourcis) internes et externes

Déclaration des éléments

La déclaration d 'une nouvelle balise se fait grâce à l 'instruction ELEMENT. La syntaxe est:

<!ELEMENT nom contenu>

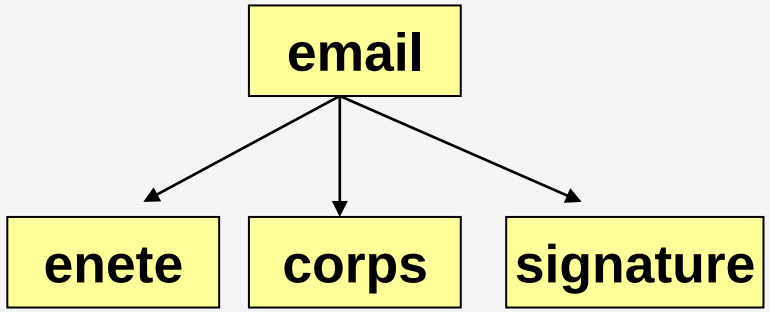
Le Contenu peut être choisi parmi cinq modèles:

- Modèle texte #PCDATA
  - exemple <!ELEMENT paragraphe (#PCDATA)>

  Pour déclarer une balise <paragraphe> qui reçoit du texte

  <paragraphe>Texte</paragraphe>
- Modèle séquence d 'éléments fils :
  - Une séquence définit les éléments fils autorisés à apparaître dans une balise.
  - Exemple : **<!ELEMENT email (entete, corps, signature)>**

```
<email>
   <entete>….</entete>
   <corps>…..</corps>
   <signature>……</signature>
</email>
```



Déclaration des éléments

- Modèle séquence d ’éléments fils :
  - Chaque nom d ’élément peut être suffixé par un indicateur d ’occurrence.
  - Cet indicateur permet de préciser le nombre d ’instances de l ’élément fils que l ’élément peut contenir. Ces indicateurs sont au nombre de 3.
    - **element?** : Indique que l ’élément peut apparaître zéro ou une fois;
    - **element+** : Indique que l ’élément peut apparaître une ou plusieurs fois;
    - **element*** : Indique l ’élément peut apparaître zéro, une ou plusieurs fois.
  - L ’absence de cet indicateur d ’occurrence indique que l ’élément peut apparaître une seule fois au sein de son élément parent.
  - On peut utiliser les parenthèses pour affecter un indicateur d ’occurrence à plusieurs élément

Déclaration des éléments

- Modèle séquence d ’éléments fils : (suite)
  - Exemple:

    <!ELEMNT livre (preface,introduction,(intro-chap,contenu-chap)+)>

  Nous pourrons alors écrire :

```
<livre>
   <preface>…..</preface>
   <introduction> …. </introduction>
   <intro-chap> …..</intro-chap>
   <contenu-chap> …. </contenu-chap>
   <intro-chap> …..</intro-chap>
   <contenu-chap> …. </contenu-chap>
</livre>
```

  - Le choix le plus libre qu ’une définition de document puisse accorder correspond à la définition suivante :

<!ELEMENT paragraphe (a | b | c)*>

Déclaration des éléments

- Modèle mixte :
  - Le modèle mixte naît de l 'utilisation conjointe de #PCDATA et d 'éléments fils dans une séquence.
  - Un tel modèle impose d 'utiliser un ordre libre et de disposer #PCDATA en première position. De plus l 'indicateur d 'occurrence doit être *.
  - Exemple : <!ELEMENT p(#PCDATA | u | i)* >
  - En complétant ces déclarations par les deux suivantes :

    <!ELEMENT u (#PCDATA)>

    <!ELEMENT i (#PCDATA)>

  La balise p peut être alors utilisée :

  ** **Dans son livre**<u>** "la huitième couleur" **</u>**, Tirry Pratchet nous raconte les aventures du premier touriste du Disque-Monde , le dénommé ****Deux Fleurs****

  ** **

Déclaration des éléments

- Modèle Vide.
  - Le mot EMPTY permet de définir une balise vide.
  - La balise
 de HTML reproduite en XML en constitue un parfait exemple :

  <!ELEMENT livre EMPTY >

  - Une balise vide ne pourra en aucun cas contenir un élément fils.
  - Un élément vide peut avoir des attributs.

  <livre id="534" titre="java" datePret="2006-11-12" rendu="oui"/>
- Modèle différent.
  - Ce dernier modèle permet de créer des balises dont le contenu est totalement libre.
  - Pour utiliser ce modèle, on utilise le mot ANY
  - Exemple : **<!ELEMENT disque ANY>**

  l 'élément disque peut être utilisé librement.

Déclaration d ’attributs

La déclaration d ’attributs dans une DTD permet de préciser quels attributs peuvent être associés à un élément donnée.

Ces attributs reçoivent de plus une valeur par défaut.

La syntaxe de déclaration d ’un attribut est :

- **<!ATTLIST nom-balise nom-attribut type-attribut présence >**

Il est possible de déclarer plusieurs attribut au sein d ’une même balise ATTLIST.

Exemples:

- **<!ELEMENT personne EMPTY>**
- **<!ATTLIST personne nom CDATA #REQUIRED >**
- **<!ATTLIST personne prenom CDATA #REQUIRED >**

OU:

- **<!ELEMENT personne EMPTY>**
- **<!ATTLIST personne**
  nom CDATA #REQUIRED
  prenom CDATA #REQUIRED>

Déclaration d ’attributs

Les types d’attributs les plus courants sont:

- Type **CDATA** : signifie que la valeur de l ’attribut doit être une chaîne de caractères;
- Type **NMTOKEN** : signifie que la valeur de l ’attribut doit être une chaîne de caractères ne contenant pas d’espaces et de caractères spéciaux;
- Type **ID** : Ce type sert à indiquer que l'attribut en question peut servir d'*identifiant* dans le fichier XML. Deux éléments ne pourront pas posséder le même attribut possédant la même valeur.
- Type énuméré : une liste de choix possible de type (A|B|C) dans laquelle A, B, C désignent des valeurs que l ’attribut pourra prendre.
  - Une valeur booléenne peut donc être représentée par en XML par les deux déclarations suivantes:

```
<!ELEMENT  boolean  EMPTY >
<!ATTLIST  boolean  value (true | false ) 'false'>
```

Déclaration d ’attributs

Le terme Présence permet de définir comment doit être gérée la valeur de l ’attribut. Il existe 4 types de présences :

- **La valeur par défaut** de l ’attribut (comme dans le cas de l ’exemple de l ’élément boolean)
- **#REQUIRED** Indique que l ’attribut doit être présent dans la balise et que sa valeur doit être obligatoirement spécifiée.
- **#IMPLIED** Indique que la présence de l ’attribut est facultative;
- **#FIXED** "valeur " Fixe la valeur de l ’attribut.

Déclaration d ’entités

Une entité définit un raccourcis vers un contenu qui peut être soit une chaîne de caractères soit le conteu d’un fichier.

La déclaration générale est : **<!ENTITY nom type>**

- Exemple d ’entité caractères: **<!ENTITY eacute "é">**

C ’est comme si je définie une constante eacaute qui représente le caractère « é ».

- Exemple d’entité chaîne de caractères:
  - **<!ENTITY ADN "Acide désoxyribonucléique">**
  - Pour utiliser cette entité, on écrit : **&ADN;**
- Exemples d ’entité relative au contenu d ’un fichier:
  - **<!ENTITY autoexec SYSTEM "file:/c:/autoexec.bat " >**
  - **<!ENTITY notes SYSTEM "../mesnotes/notes.txt" >**

Pour utiliser une entité dans un document XML, on écrit: **&nom-entité;**

Exemple 1

Un opérateur Télécom fournit périodiquement, à l'opérateur de réglementation des télécoms ANRT un fichier XML qui contient les clients de cet opérateur. Chaque client possède plusieurs abonnements et chaque abonnement reçoit plusieurs factures. Un exemple de fichier XML correspondant à ce problème est le suivant :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<operateur>
   <client code="1" nom="hassouni">
      <abonnement num="123" type="GSM" dateAb="2006-1-11">
         <facture numFact="432" dateFact="2006-2-11" montant="350.44" reglee="oui"/>
         <facture numFact="5342" dateFact="2006-3-11" montant="450" reglee="oui"/>
         <facture numFact="5362" dateFact="2006-4-13" montant="800.54" reglee="non"/>
      </abonnement>
      <abonnement num="2345" type="FIXE" dateAb="2006-12-1">
         <facture numFact="5643" dateFact="2007-1-12" montant="299" reglee="oui"/>
         <facture numFact="6432" dateFact="2007-2-12" montant="555" reglee="non"/>
      </abonnement>
   </client>
   <client code="2" nom="abbassi">
      <abonnement num="7543" dateAb="2006-12-1" type="GSM">
         <facture numFact="7658" dateFact="2007-2-12" montant="350.44" reglee="oui"/>
         <facture numFact="7846" dateFact="2007-3-12" montant="770" reglee="non"/>
      </abonnement>
   </client>
</operateur>
```

Travail demandé

1. Faire une représentation graphique de l'arbre XML.
2. Ecrire une DTD qui permet de valider ce document XML
3. Créer le fichier XML
4. Ecrire une feuille de style XSL qui permet de transformer le document XML en une page HTML qui permet d'afficher pour chaque client la liste de ses abonnement en affichant le montant total des factures de chaque abonnement

Nom du client:hassouni

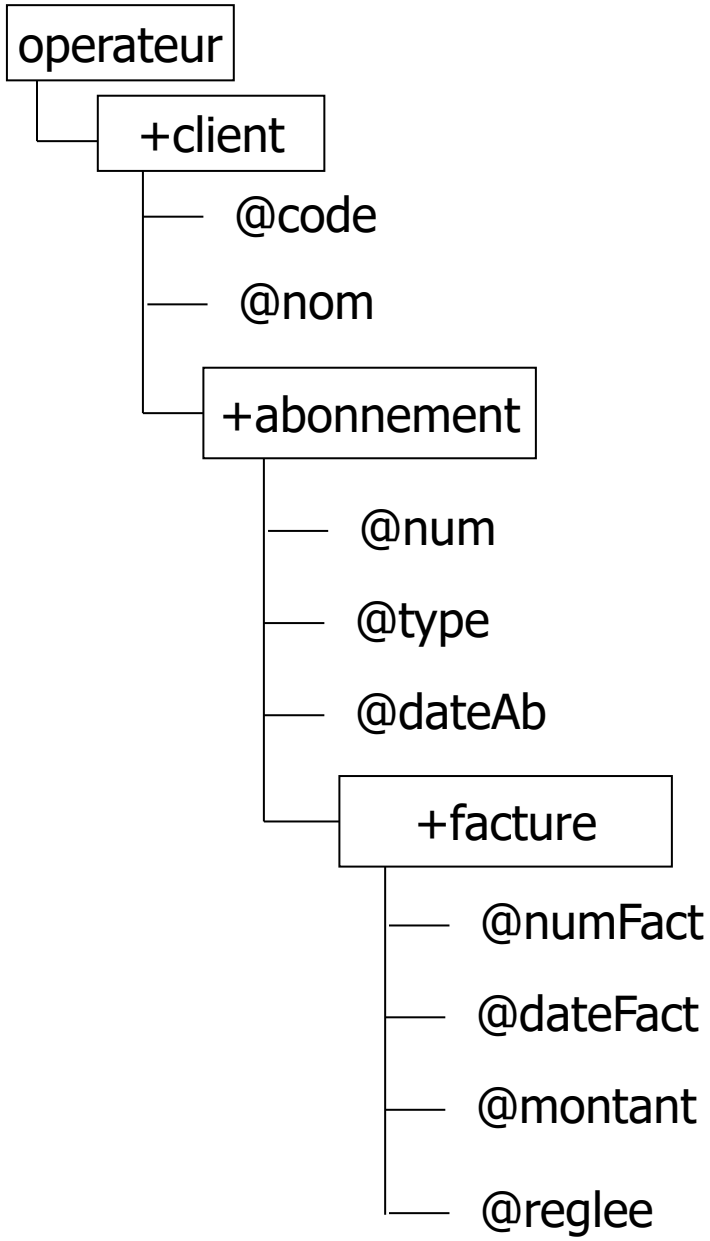
Num Abonnement	Type	Date Abonnement	Montant Total des factures
123	GSM	2006-1-11	1600.98
2345	FIXE	2006-12-1	854

Nom du client:abbassi

Num Abonnement	Type	Date Abonnement	Montant Total des factures
7543	GSM	2006-12-1	1120.44

Correction : DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT operateur (client+) >
<!ELEMENT client (abonnement+) >
<!ELEMENT abonnement (facture+) >
<!ELEMENT facture EMPTY >
<!ATTLIST client
          code NMTOKEN #REQUIRED
          nom CDATA #REQUIRED>
<!ATTLIST abonnement
          num NMTOKEN #REQUIRED
          type (GSM|FIXE) 'FIXE'
          dateAb CDATA #REQUIRED>
<!ATTLIST facture
          numFact NMTOKEN #REQUIRED
          dateFact CDATA #REQUIRED
          montant CDATA #REQUIRED
          reglee (oui|non) 'non'>
```



Corrigé : XSL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="/">
   <html>
     <head></head>
     <body>
       <xsl:for-each select="operateur/client">
         <h3>
           Nom Client : <xsl:value-of select="@nom"/>
         </h3>
       <table border="1" width="80%">
         <tr>
           <th>Num</th><th>Type</th><th>Date</th><th>Total Factures</th>
         </tr>
         <xsl:for-each select="abonnement">
           <tr>
             <td><xsl:value-of select="@num"/></td>
             <td><xsl:value-of select="@type"/></td>
             <td><xsl:value-of select="@dateAb"/></td>
             <td><xsl:value-of select="sum(facture/@montant)"/></td>
           </tr>
         </xsl:for-each>
       </table>
       </xsl:for-each>
     </body>
   </html>
 </xsl:template>
</xsl:stylesheet>
```

Application

Nous souhaitons concevoir un format de fichier XML échangés entre la bourse et ses différents partenaires.
Un exemple de fichier XML manipulé est le suivant :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bourse>
   <societe type="banque">
      <codeSociete>SGMB</codeSociete>
      <nomSociete>Société Générale</nomSociete>
      <datIntroduction>2000-11-01</datIntroduction>
      <cotations>
         <cotation num="23" dateCotation="2010-01-01" valeurAction="650"/>
         <cotation num="24" dateCotation="2010-01-02" valeurAction="680"/>
      </cotations>
   </societe>
   <societe type="assurance">
      .....
      .....
   </societe>
</bourse>
```

Exemple d'application

Questions :

1. Faire une représentation graphique de l'arbre XML
2. Ecrire la DTD qui permet de déclarer la structure du document XML. (on suppose que le fichier XML contient deux de types de sociétés « banque» et « assurance»)
3. Ecrire un schéma XML équivalent.
4. Ecrire une feuille de style qui permet d'afficher les cotations des sociétés de type « banque».

Cotations des sociétés de type Banque :

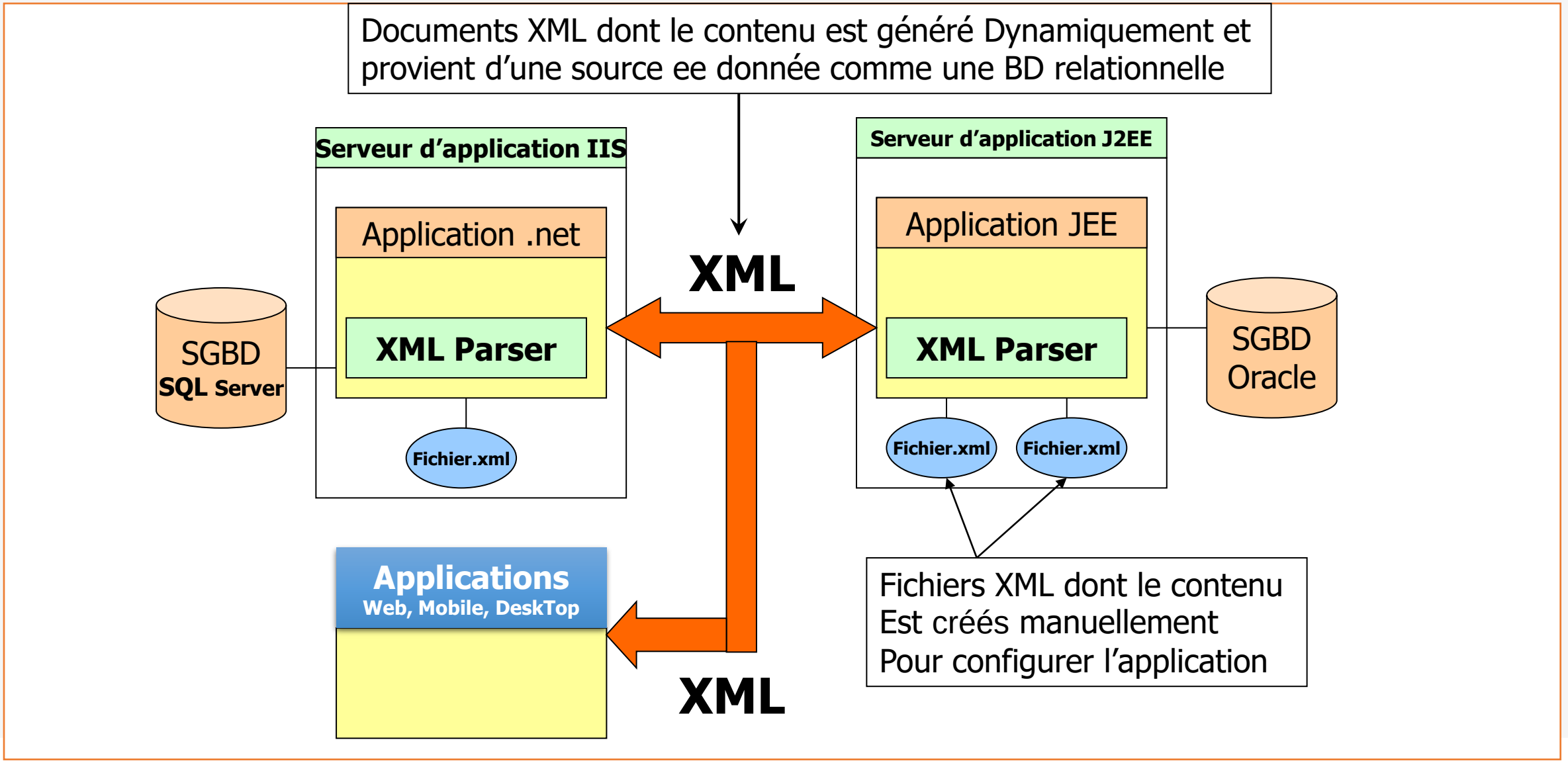
- CODE SOCIETE : SGMB
- NOM SOCIETE : Société Générale

Cotations :

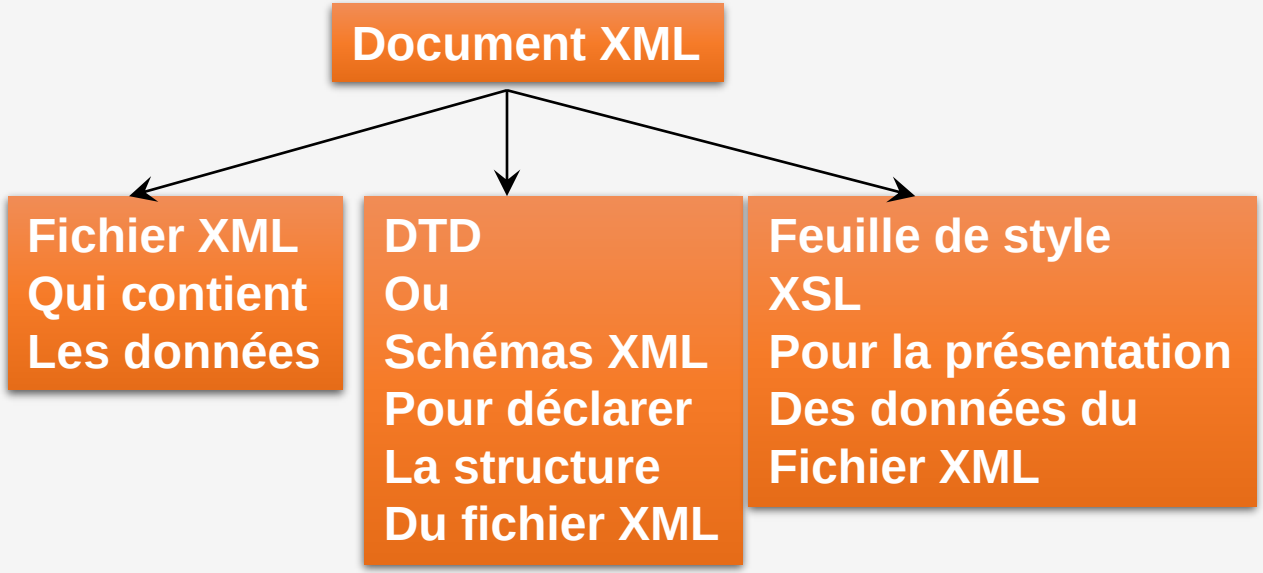
Num cotation	Date Cotation	VAL ACTION
23	2010-01-01	650
24	2010-01-02	680
25	2010-01-03	720
Moyenne des cotations		683.3333333333334

- Nombre total de sociétés :2
- Nombre de sociétés de type banque :1
- Nombre total de sociétés de type assurance :1

XML pour l'échange de données entre les Systèmes d'Information

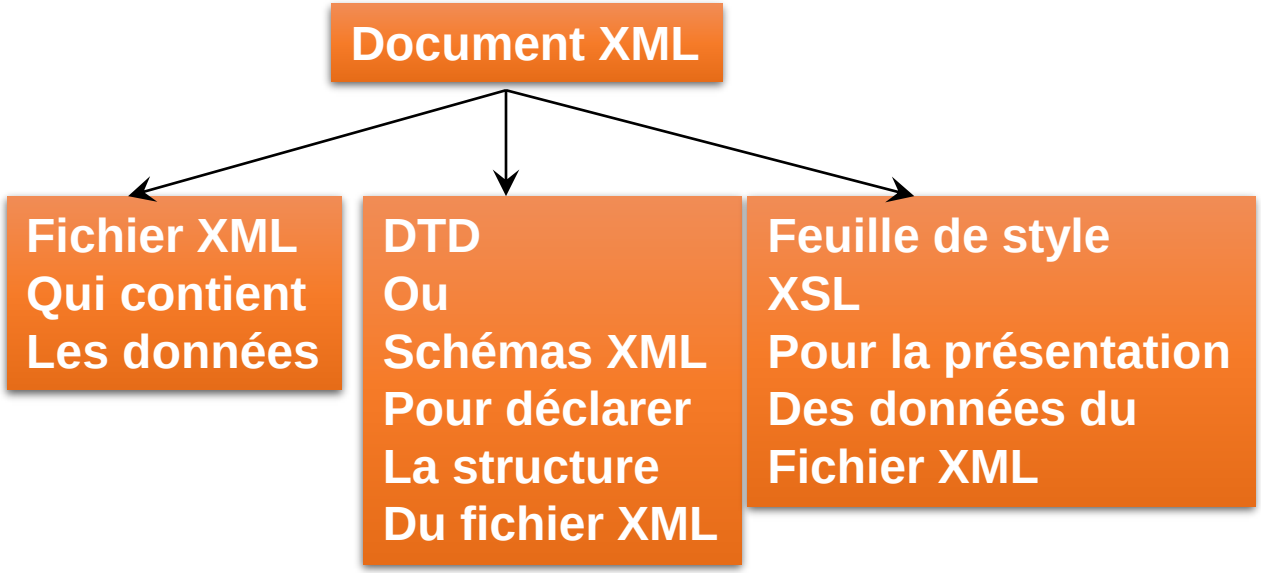


Schémas XML


Document XML
Fichier XML
Qui contient
Les données
DTD
Ou
Schémas XML
Pour déclarer
La structure
Du fichier XML
Feuille de style
XSL
Pour la présentation
Des données du
Fichier XML

Structure d'un document XML

Un document XML se compose de 3 fichiers :



- Le fichier XML stocke les données du document sous forme d'un arbre
- DTD (Data Type Definition) ou Schémas XML définit la structure du fichier XML
- La feuille de style définit la mise en forme des données de la feuille xml

Apports des schémas

Conçu pour pallier aux déficiences pré citées des DTD, XML Schema propose, en plus des fonctionnalités fournies par les DTD, des nouveautés :

- Le typage des données est introduit, ce qui permet la gestion de booléens, d'entiers, d'intervalles de temps... Il est même possible de créer de nouveaux types à partir de types existants.
- La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément.
- Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif.
- Le support des espaces de nom.
- Les Schémas XML sont des documents XML ce qui signifie d’un parseur XML permet de les manipuler facilement.

Structure de base d'un schéma xml

Comme tout document XML, un Schema XML commence par un prologue, et a un élément racine.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

      <!-- déclarations d'éléments, d'attributs et de types ici -->

</xsd:schema>
```

L'élément racine est l'élément **xsd:schema**.

Les éléments du XML schéma sont définis dans l'espace nom `http://www.w3.org/2000/10/XMLSchema` qui est préfixé par xsd.

Cela signifie que tous le éléments de XML schéma commencent par le préfix xsd (<xsd:element>)

Déclarations d'éléments

Un élément, dans un schéma, se déclare avec la balise **<xsd:element>**.

- Par exemple,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="remarque" type="xsd:string"> </xsd:element>
  <xsd:element name="contacts" type="typeContacts"> </xsd:element>
  <!-- déclarations de types ici -->
</xsd:schema>
```

- remarque est un élément de type simple
- contacts est un élément de type complexe

Déclarations d'attributs

Un attribut, dans un schéma, se déclare avec la balise **<xsd:attribute>**.

Un attribut ne peut être que de type simple

Exemple:

```
<xsd:element name="contacts" type="typeContacts"/>
<xsd:element name="remarque" type="xsd:string"/>
  <!-- déclarations de types ici -->
<xsd:complexType name="typeContacts">
   <!-- déclarations du modèle de contenu ici -->
 <xsd:attribute name="maj" type="xsd:date" />
</xsd:complexType>
```

Ici on déclare que contacts est un élément de type complexe et qu'il possède un attribut maj de type date

Contraintes d'occurrences pour les attribus

L'élément **attribute** d'un Schema XML peut avoir trois attributs optionnels :

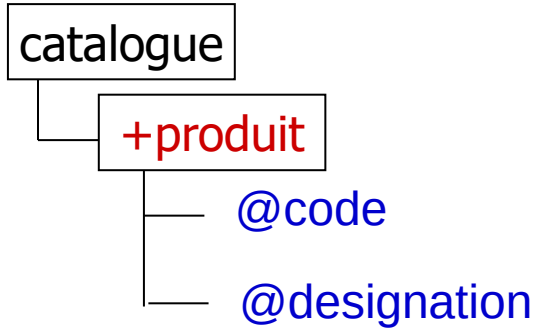
- **use** : indique le présence , il peut prendre pour valeur required(obligatoire), optional(facultatif) ou prohibited (ne doit pas apparaitre)
- **default** : pour indiquer la valeur par défaut
- **fixed** : indique l'attribut est renseigné, la seule valeur que peut prendre l'attribut déclaré est celle de l'attribut **fixed**. Cet attribut permet de "réserver" des noms d'attributs pour une utilisation future, dans le cadre d'une mise à jour du schéma.

Exemple :

```
<xsd:attribute name="maj" type="xsd:date" use="optional"  default="2003-10-11" />
```

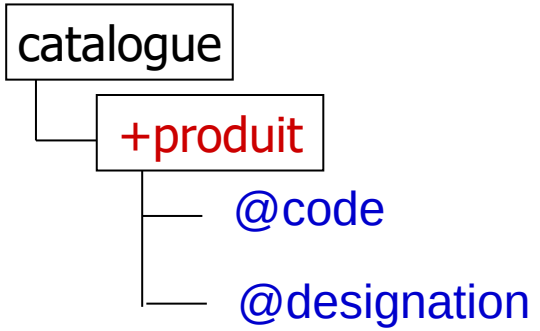
Premier Exemple schéma xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:element name="catalogue">
    <xsd:complexType>
       <xsd:sequence>
          <xsd:element name="produit" type="T_PRODUIT" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
 </xsd:element>

 <xsd:complexType name="T_PRODUIT">
      <xsd:attribute name="code" type="xsd:int" use="required"/>>
      <xsd:attribute name="designation" type="xsd:string" use="required"/>
   </xsd:complexType>
</xsd:schema>
```



Fichier XML respectant le schéma précédent

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file:/C:/m/catalogue.xsd">
   <produit code="1" designation="PC HP 650"/>
   <produit code="2" designation="Imprimante HP 690"/>
</catalogue>
```



Déclaration d'élément ne contenant que du texte avec un (ou plusieurs) attribut(s)

- Un tel élément est de type complexe, car il contient au moins un attribut.
- Afin de spécifier qu'il peut contenir également du texte, on utilise l'attribut **mixed** de l'élément **<xsd:complexType>**.
- Par défaut, **mixed="false"**; il faut dans ce cas forcer **mixed="true"**.
- Par exemple:

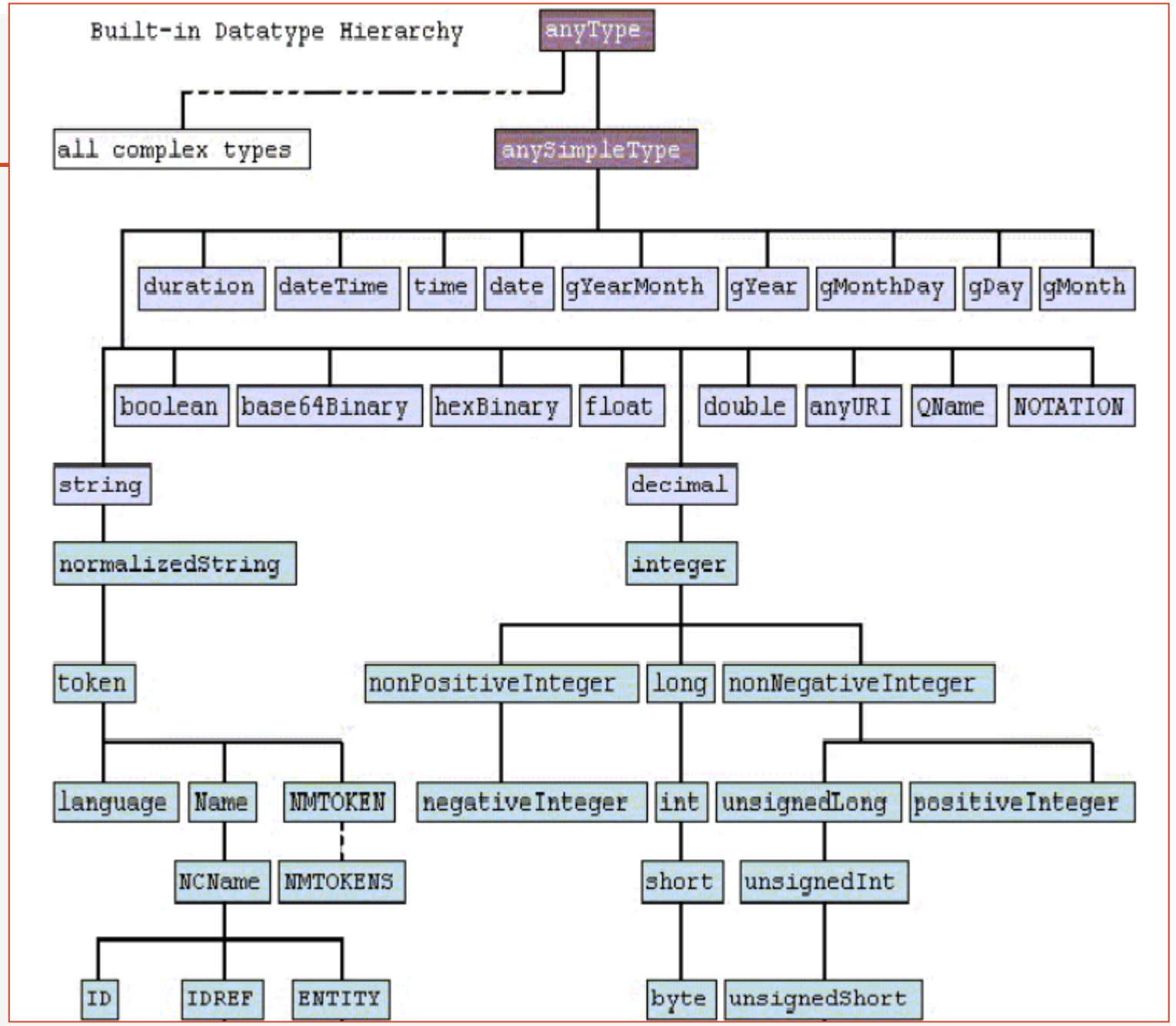
```
<xsd:element name="elt">
 <xsd:complexType mixed="true">
  <xsd:attribute name="attr" type="xsd:string" use="optional" />
 </xsd:complexType>
</xsd:element>
```

Déclaration et référencement

- Il est beaucoup plus avantageux, pour des raisons de clarté, d'ordonner ces déclarations, ainsi qu'on peut le voir sur cet exemple:

```
<xsd:element name="pages" type="xsd:positiveInteger"/>

<xsd:element name="auteur" type="xsd:string"/>

<xsd:element name="livre">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="auteur" />
      <xsd:element ref="pages" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Types Atomiques



Types simples : Listes

- Les types listes sont des suites de types simples (ou *atomiques*).
- Il est possible de créer une liste personnalisée, par "dérivation" de types existants. Par exemple,

```
<xsd:simpleType name="numeroDeTelephone">
  <xsd:list itemType="xsd:unsignedByte" />
</xsd:simpleType>
```

- Un élément conforme à cette déclaration serait

<telephone>01 44 27 60 11</telephone>.

- Il est également possible d'indiquer des contraintes plus fortes sur les types simples ; ces contraintes s'appellent des "facettes".

Elles permettent par exemple de limiter la longueur de notre numéro de téléphone à 10 nombres.

Types simples: Unions

- Les listes et les types simples intégrés ne permettent pas de choisir le type de contenu d'un élément.
- On peut désirer, par exemple, qu'un type autorise soit un nombre, soit une chaîne de caractères particuliers.
- Il est possible de le faire à l'aide d'une déclaration d'union.
- Par exemple, sous réserve que le type simple numéroDeTéléphone ait été préalablement défini (voir précédemment), on peut déclarer...

```
<xsd:simpleType name="numeroTelTechnique">
 <xsd:union memberTypes="xsd:string numeroDeTelephone" />
</xsd:simpleType>
```

- Les éléments suivants sont alors des "instances" valides de cette déclaration :

<téléphone>18**</téléphone>**

<téléphone>Pompiers**</téléphone>**

Types complexes

- Un élément de type simple ne peut contenir de sous-élément ou des attributs.
- Un type complexe est un type qui contient soit des éléments fils, ou des attributs ou les deux.
- On peut alors déclarer,
  - Des séquences d'éléments,
  - Des types de choix

Types complexes: séquences d'éléments:

- Nous savons déjà comment, dans une DTD, nous pouvons déclarer un élément comme pouvant contenir une suite de sous-éléments, dans un ordre déterminé.
- Il est bien sûr possible de faire de même avec un schéma.
- On utilise pour ce faire l'élément **xsd:sequence**, qui reproduit l'opérateur, du langage DTD. Exemple:

```
<xsd:complexType>
 <xsd:sequence>
  <xsd:element name="nom" type="xsd:string" />
  <xsd:element name="prénom" type="xsd:string" />
  <xsd:element name="dateDeNaissance" type="xsd:date"/>
  <xsd:element name="adresse" type="xsd:string" />
  <xsd:element name="email" type="xsd:string" />
  <xsd:element name="tel" type="numéroDeTéléphone" />
 </xsd:sequence>
</xsd:complexType>
```

Types Complexes : Choix d'élément

On peut vouloir modifier la déclaration de type précédente en stipulant qu'on doive indiquer soit l'adresse d'une personne, soit son adresse électronique. Pour cela, il suffit d'utiliser un élément **xsd:choice** :

```
<xsd:complexType name="typePersonne">
  <sequence>
   <xsd:element name="nom" type="xsd:string" />
   <xsd:element name="prénom" type="xsd:string" />
   <xsd:element name="dateDeNaissance" type="xsd:date" />
   <xsd:choice>
     <xsd:element name="adresse" type="xsd:string" />
     <xsd:element name="email" type="xsd:string" />
   </xsd:choice>
   <xsd:element name="tel" type="numéroDeTéléphone" />
  </sequence>
</xsd:complexType>
```

Types Complexe : Elément All

L'élément All indique que les éléments enfants doivent apparaître une fois (ou pas du tout), et dans n'importe quel ordre.

Cet élément **xsd:all** doit être un enfant direct de l'élément **xsd:complexType**.

Par exemple

```
<xsd:complexType>
  <xsd:all>
    <xsd:element name="nom" type="xsd:string" />
    <xsd:element name="prénom" type="xsd:string" />
    <xsd:element name="dateDeNaissance" type="xsd:date" />
    <xsd:element name="adresse" type="xsd:string" />
    <xsd:element name="adresseElectronique" type="xsd:string" />
    <xsd:element name="téléphone" type="numéroDeTéléphone" />
  </xsd:all>
</xsd:complexType>
```

Indicateur d'occurrences

- Pendant la déclaration d'un élément, on peut indiquer le nombre minimum et le nombre maximum de fois qu'il doit apparaître
- On utilise pour cela les attributs **minOccurs** et **maxOccurs**:
  - **minOccurs** prend par défaut la valeur 1 et peut prendre les autres valeurs positives
  - **maxOccurs** prend par défaut la valeur 1 et peut prendre les autres valeurs positive ou unbounded (infini).

Dérivation

- Les types simples et complexes permettent déjà de faire plus de choses que les déclarations dans le langage DTD.
- Il est possible de raffiner leur déclaration de telle manière qu'ils soient
  - une "**restriction**"
  - ou une **extension** d'un type déjà existant, en vue de préciser un peu plus leur forme.

Dérivation : Restriction de types

- Une "facette" permet de placer une contrainte sur l'ensemble des valeurs que peut prendre un type de base.
- Par exemple, on peut souhaiter créer un type simple, appelé MonEntier, limité aux valeurs comprises entre 0 et 99 inclus.
- On dérive ce type à partir du type simple prédéfini **nonNegativeInteger**, en utilisant la facette **maxExclusive**.

```
<xsd:simpleType name="monEntier">
  <xsd:restriction base="nonNegativeInteger">
    <xsd:maxExclusive value="100" />
  </xsd:restriction>
</xsd:simpleType>
```

Dérivation : Restriction de types

Il existe un nombre important de facettes qui permettent de :

- fixer, restreindre ou augmenter la longueur minimale ou maximale d'un type simple
- énumérer toutes les valeurs possibles d'un type
- prendre en compte des expressions régulières
- fixer la valeur minimale ou maximale d'un type (voir l'exemple ci-dessus)
- fixer la précision du type...

Dérivation : Exemples de facettes

- Limiter les différentes valeurs possible d'un type:

```
<xsd:attribute name="jour" type="jourSemaine" use="required" />
<xsd:simpleType name="jourSemaine">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="lundi" />
    <xsd:enumeration value="mardi" />
    <xsd:enumeration value="mercredi" />
    <xsd:enumeration value="jeudi" />
    <xsd:enumeration value="vendredi" />
    <xsd:enumeration value="samedi" />
    <xsd:enumeration value="dimanche" />
  </xsd:restriction>
</xsd:simpleType>
```

Dérivation : Exemples de facettes

Limiter la longueur d'une chaîne de caractères:

```
<xsd:simpleType name="monType">
  <xsd:restriction base="xsd:string">
  <xsd:length value="21" />
  </xsd:restriction>
</xsd:simpleType>
```

Dérivation : Exemples de facettes

- Expressions régulières pour une adresse email:

```
<xsd:simpleType name="typeAdresseElectronique">
 <xsd:restriction base="xsd:string">
    <xsd:pattern value="(.)+@(.)+" />
  </xsd:restriction>
</xsd:simpleType>
```

Application

- Un service d'envoi des mandats établit mensuellement un rapport qui contient les mandats envoyés.
- Chaque mandat concerne un expéditeur et un destinataire. L'expéditeur est défini par son cin, son nom, son prénom et sa ville.
- Le destinataire est défini également par son cin, son nom, son prénom et sa ville.

```
<?xml version="1.0" encoding="UTF-8"?>
<rapport>
   <mandat num="7124536" date="2007-1-1" montant="1000" etat="reçu">
      <expediteur cin="A123245" nom="slimani" prenom="youssef" ville="casa"/>
      <destinataire cin="P98654" nom="hassouni" prenom="laila" ville="fes"/>
   </mandat>
   <mandat num="7124537" date="2007-1-1" montant="3200" etat="non reçu">
      <expediteur cin="M123245" nom="alaoui" prenom="mohamed" ville="marrakech"/>
      <destinataire cin="M92654" nom="alaoui" prenom="imane" ville="casa"/>
   </mandat>
   <mandat num="7124538" date="2007-1-2" montant="500" etat="reçu">
      <expediteur cin="H123222" nom="qasmi" prenom="slim" ville="oujda"/>
      <destinataire cin="B91154" nom="qasmi" prenom="hassan" ville="rabat"/>
   </mandat>
</rapport>
```

Travail à faire

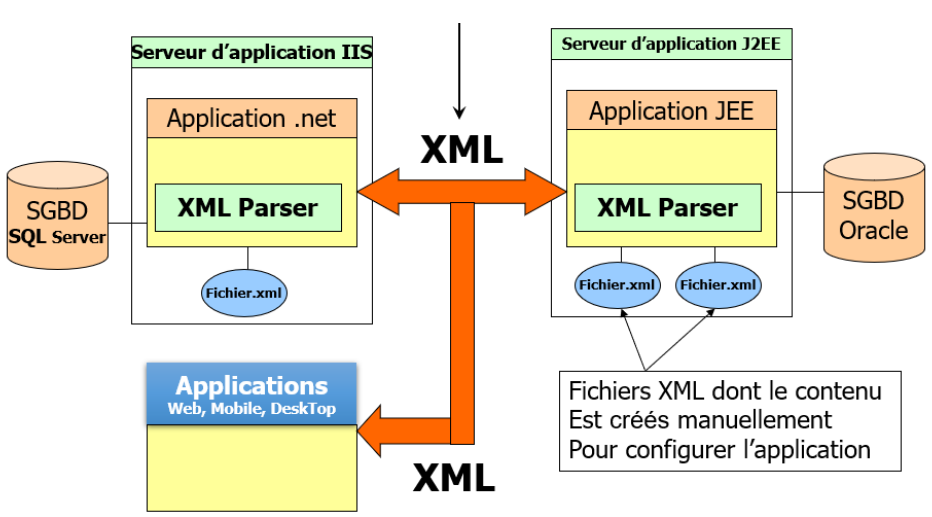
1. Faire une représentation graphique de l'arbre XML
2. Ecrire une DTD qui permet de déclarer la structure du document XML
3. Ecrire le schéma XML qui permet de déclarer la structure du document XML
4. Créer le fichier XML valide respectant ce schéma.
5. Ecrire une feuille de style XSL qui permet de transformer le document XML en document HTML suivant :

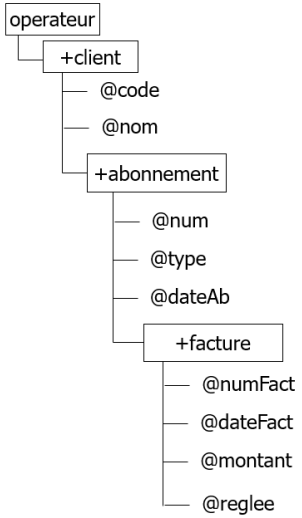
Num Mandat	Date	Expéditeur	Destinataire	Etat	Montant
7124536	2007-1-1	• CIN: A123245 • Nom:slimani	• CIN: P98654 • Nom:hassouni	reçu	1000
7124537	2007-1-1	• CIN: M123245 • Nom:alaoui	• CIN: M92654 • Nom:alaoui	non reçu	3200
7124538	2007-1-2	• CIN: H123222 • Nom:qasmi	• CIN: B91154 • Nom:qasmi	reçu	500
				Total des mandats	4700
				Total des mandats reçus	1500
				Total des mandats non reçus	3200

SVG : Scalable Vector Graphics

SVG : Scalable Vector Graphics

Master Ingénierie Informatique, Big Data et Cloud Computing II-BDCC, ENSET Mohammedia , Université Hassan II de Casablanca






```
ml version="1.0" encoding="UTF-8"?>
erateur>
 <client code="1" nom="hassouni">
     <abonnement num="123" type="GSM" dateAb="2006-1-11">
         <facture numFact="432" dateFact="2006-2-11" montant="350.44" reglee="oui"/>
         <facture numFact="5342" dateFact="2006-3-11" montant="450" reglee="oui"/>
         <facture numFact="5362" dateFact="2006-4-13" montant="800.54" reglee="non"/>
     </abonnement>
     <abonnement num="2345" type="FIXE" dateAb="2006-12-1">
         <facture numFact="5643" dateFact="2007-1-12" montant="299" reglee="oui"/>
         <facture numFact="6432" dateFact="2007-2-12" montant="555" reglee="non"/>
     </abonnement>
 </client>
 <client code="2" nom="abbassi">
     <abonnement num="7543" dateAb="2006-12-1" type="GSM">
         <facture numFact="7658" dateFact="2007-2-12" montant="350.44" reglee="oui"/>
         <facture numFact="7846" dateFact="2007-3-12" montant="770" reglee="non"/>
     </abonnement>
 </client>
perateur>
```


Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Introduction

- SVG : Scalable Vector Graphics. Ce langage permet d'écrire des graphiques vectoriels 2D en XML.
- Il a été inventé en 1998 par un groupe de travail (comprenant Microsoft, Autodesk, Adobe, IBM, Sun, Netscape, Xerox, Apple, Corel, HP, ILOG...) pour répondre à un besoin de graphiques légers, dynamiques et interactifs.
- Une première ébauche du langage est sortie en octobre 1998 et en juin 2000 apparaît la première version du **Viewer Adobe** (*plugin* permettant de visualiser le SVG).
- Le SVG s'est très vite placé comme un concurrent de **Flash** et à ce titre, Adobe a intégré ce langage dans la plupart de ses éditeurs (dont les principaux sont **Illustrator** et **Golive**).

Pourqoui SVG

Les raisons pouvant pousser à l'adoption d'un format comme SVG sont nombreuses ;

- Liées aux avantages du graphisme vectoriel :
  - Adaptation de l'affichage à des media variés et à des tailles différentes ;
  - Taille de l'image après compression est très faible ;
  - ....
- Liées aux avantages particuliers du format SVG :
  - Insertion dans le monde XML/XHTML :
  - Modèle de couleurs sophistiqué, filtres graphiques
  - Possibilité d'indexage par les moteurs de recherche;
  - Possibilité de partager du code ;
  - Meilleures capacités graphiques dans l'ensemble.

Outils

Edition

- Comme SVG est un format XML, n'importe quel éditeur de texte suffit.
- Il est cependant possible d'utiliser un éditeur dédié comme **WebDraw** de Jasc, ou InkScape. **Adobe Illustrator**, **Corel Draw** permettent aussi d'exporter au format SVG.

Visualisation

- SVG n'est actuellement pas supporté en natif par tous les navigateurs. On distingue plusieurs degrés d'avancement :
- Internet Explorer ne le supporte pas en natif. Il est donc nécessaire d'installer un *plugin*. le *plugin* **SVG Viewer 3.03**, téléchargeable gratuitement sur le site d'Adobe : http://www.adobe.com/svg/
- FireFox, à partir de sa version 1.5, a entrepris de supporter en natif le format SVG,
- Opera a commencé à prendre en charge le SVG dans sa version 8.0. Le support SVG d'Opera 8.5 inclut maintenant des animations.
- Google Chrome

Structure d'un fichier SVG

A- Prologue:

Un fichier SVG commence par une déclaration de version XML standard, comme par exemple

```
<?xml  version="1.0“ encoding="ISO-8859-1" ?>
```

- **Il faut alors ajouter le "DocType" correspondant à la version SVG utilisée ;**

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```

- **Il est cependant préférable, en cours de développement, de copier localement la DTD de SVG, et d'y accéder en local :**

```
<!DOCTYPE svg SYSTEM "svg10.dtd">
```

Structure d'un fichier SVG

B. Elément racine

- **La racine d'un fichier SVG est un élément <svg>.**
- **Il est nécessaire de définir deux "espaces de noms", un par défaut et un second permettant d'avoir accès à d'autres fonctionnalités que nous verrons par la suite, comme suit ;**

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:link="http://www.w3.org/1999/xlink">
    (...)
</svg>
```

- **La taille de la fenêtre SVG est définie par les attributs width et height de l'élément svg**

```
<svg width="400" height="250" xmlns="http://www.w3.org/2000/svg">
```

Structure d'un fichier SVG

Imbrication d'un fichier SVG dans HTML

- La version canonique (déconseillée) demande d'utiliser la balise <embed>, sous la forme

```
<embed src="..." width="..." height="..." type="image/svg+xml">
```

- La solution la plus souple d'emploi reste d'utiliser un environnement <iframe>. Par exemple :

```
<iframe src="fichier.svg" height="540" width="95%" frameborder="0">
  <p>(Contenu alternatif: image+texte, texte seulement...)</p>
</iframe>
```

Eléments graphiques de base

SVG définit un certain nombre d'éléments graphiques de base. Voici la liste des éléments les plus importants :

- **Texte avec <text> ;**
- **Rectangles <rect> ;**
- **Le cercle <circle> et l'ellipse <ellipse> ;**
- **Lignes <line> et poli-lignes <polyline> ;**
- **Polygones <polygon>;**
- **Formes arbitraires avec <path>.**

Le rendu

SVG définit quelques dizaines d'attributs-propriétés applicables à certains éléments. En ce qui concerne les éléments graphiques, voici les deux plus importants :

- **stroke, définit la forme du bord d'un objet ;**
- **fill, définit comment le contenu d'un objet est rempli.**

SVG possède deux syntaxes différentes pour définir la mise en forme d'un élément :

- **L'attribut style reprend la syntaxe et les styles de CSS2 ;**
- **Pour chaque style, il existe aussi un attribut de présentation SVG, cela simplifie la génération de contenus SVG avec XSLT.**

Exemple :

```
<rect x="200" y="100" width="60" height="30" fill="red" stroke="blue" stroke-width="3" />
```

Le code précédent a le même effet que ;

```
<rect x="200" y="100" width="60" height="30" style="fill:red;stroke:blue;stroke-width:3" />
```

Le rendu : Propriété fill

- fill permet de gérer le remplissage de l'élément. Ses propriétés sont :
  - la **couleur**, avec les mêmes conventions de représentation qu'en CSS (exemple précédent : fill="red").
  - l'**URI** d'une couleur, d'un gradient de couleur (pour obtenir un effet de dégradé) ou d'un motif de remplissage.
  - une valeur d'opacité (**opacity**), comprise entre 0 et 1.

Le rendu : Propriété stroke

- stroke permet de gérer l'entourage d'un élément de dessin. Ses propriétés sont :
  - la couleur, avec les mêmes conventions de représentation qu'en CSS
  - l'uri d'une couleur, d'un gradient de couleur (pour obtenir un effet de dégradé) ou d'un motif de remplissage.
  - une valeur d'opacité (opacity), comprise entre 0 et 1.
  - l'épaisseur (width) du trait ;
  - la jonction de ligne (linejoin)
  - la forme des angles (linecap) qui peut être butt(les lignes s'arrêtent brutalement à leur fin), round ou square (des carrés sont tracés en bout de chaque ligne).

Figures géométriques (Rectangle)

- L'élément <rect> permet de définir des rectangles.
- Les attributs de base sont :
  - x et y, qui donnent la position du coin supérieur gauche.
  - width et height, qui permettent de définir respectivement largeur et hauteur du rectangle.
  - rx et ry, qui sont les axes x et y de l'ellipse utilisée pour arrondir ; les nombres négatifs sont interdits, et on ne peut dépasser la moitié de la largeur (longueur) du rectangle.

Exemple:

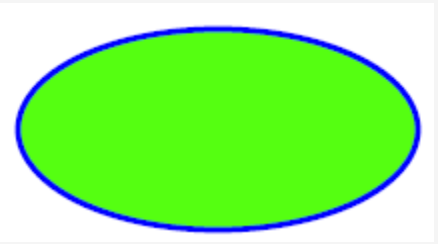
```
<rect width="100" height="50" stroke="blue" stroke-width="2"  fill="yellow" x="100" y="20" rx="10" ry="10"/>
```

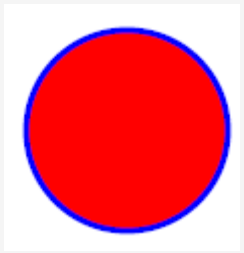
Figures géométriques (Cercle)

Un cercle est créé par l'élément <circle> et une ellipse par l'élément... <ellipse>.

Leurs attributs sont :

- cx et cy qui définissent les coordonnées du centre.
- r qui définit le rayon du cercle.
- rx et ry qui définissent les demi-axes x et y de l'ellipse.





Exemples:

```
<circle r="50" cx="300" cy="300" fill="red" stroke="blue" stroke-width="3"/>
<ellipse rx="100" ry="50" cx="350" cy="60" fill="#55FF11"/>
```

Figures géométriques (Lignes)

Une ligne est définie avec l'élément <line>,

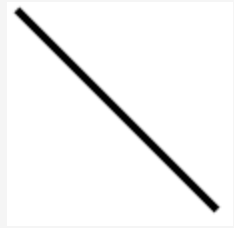
une poly-ligne par l'élément <polyline>.

Les attributs de <line> sont :

- x1 et y1, qui définissent les coordonnées du point de départ.
- x2 et y2, qui définissent les coordonnées du point d'arrivée.

L'attribut de base de <polyline> est :

- points, qui prend comme valeur une suite de coordonnées.



Exemples:

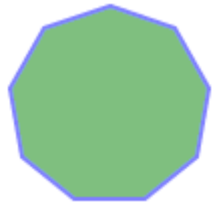
```
<line x1="150" y1="350" x2="250" y2="450" stroke="black" stroke-width="4"/>
<polyline points="150,160,250,200,300,130,350,200" fill="none"/>
```

Figures géométriques (Polygones)

Un polygone est une courbe fermée,

Une poly-ligne une courbe ouverte.

L'élément permettant de définir un polygone est <polygon>.



L'attribut de base de cet élément est:

- points, qui s'utilise de la même manière qu'avec une polyligne.

Exemples:

```
<polygon
points="350,175,382,186,399,216,393,250,367,271,332,271,306,250,300,216,317,186,350,175“ opacity="0.5"
fill="green" stroke="blue"/>
```

Formes arbitraires (Path)

a. Introduction

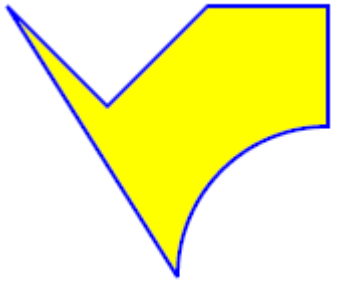
- L'élément <path> permet de définir des formes arbitraires. Ces formes peuvent avoir un contour et servir de support pour d'autres éléments

b. Attributs de base

- Ces attributs sont au nombre de 2 :
  - d, au nom peu explicite, sert à définir les *path data*, autrement dit la liste de commande permettant de tracer le chemin.
  - nominalLength, facultatif, permet de définir éventuellement la longueur totale du chemin.

Formes arbitraires (Path)

c. Path data



Les *path data* suivent les règles suivantes :

- **Toutes les instructions peuvent être abrégées en un seul caractère**
- **Les espaces peuvent être éliminés**
- **On peut omettre de répéter une commande**
- **Il existe toujours deux versions des commandes :**
  - **en majuscules : coordonnées absolues**
  - **en minuscules : coordonnées relatives**

Ces règles visent à diminuer au maximum la taille requise par la description des chemins. Les commandes sont :

- **M ou m : (*moveto*) : x,y démarre un nouveau sous-chemin**
- **Z ou z : (*closepath*) ferme un sous-chemin en traçant une ligne droite entre le point courant et le dernier moveto**
- **L ou l : (*lineto*) : x , y trace une ligne droite entre le point courant et le point (x,y).**
- **H ou h : (*horizontal lineto*) : x trace une ligne horizontale entre le point courant et le point (x,y0).**
- **V ou v : (*vertical lineto*) : y trace une ligne verticale entre le point courant et le point (x0,y).**

Il existe également des commandes permettant de tracer des courbes (courbes de Bézier, arcs…).

Exemple:

```
<path d="M 400 400 l 50 50 l 50 -50 h 60 v60 a 80,80 0 0,0 -75,75 z" id="chemin1"></path>
```

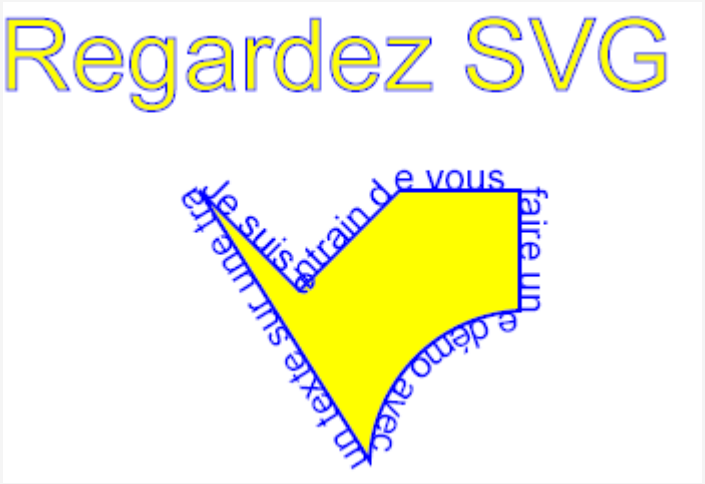
Texte

A- Balise **<text>**

- Elle possède deux attributs :
  - x et y.

c. Lien avec les chemins

- Il est possible d'écrire un texte le long d'un chemin (*path*) défini par ailleurs, par un élément **<path>** en appelant un élément <textPath>.



Exemples:

```
<text x="300" y="350" font-size="50" fill="yellow" stroke="blue" stroke-width="1">Regardez SVG</text>
<text stroke="none" fill="blue" font-size="19">
        <textPath xlink:href="#chemin1">Je suis entrain de vous faire une démo avec un texte sur une trajectoire      </textPath>
 </text>
```

Structuration: Eléments de groupages et références

- Le fragment d'un document SVG : **<svg>** ;
- Groupage d'éléments avec **<g>** ;
- Objets abstraits **<symbol>** ;
- Section de définition **<def>** ;
- Utilisation d'éléments **<use>** ;
- Titre **<title>** et description **<desc>**.

Structuration: Eléments de groupages et références

1- Le fragment d'un document SVG: <svg>

- <svg> est la racine d'un graphisme SVG. Mais on peut aussi imbriquer des éléments svg parmi d'autres et les positionner.
- Chaque <svg> crée un nouveau système de coordonnées. Ainsi on peut facilement réutiliser des fragments graphiques sans devoir modifier des coordonnées.

2- Groupage d'éléments avec <g>

- Cet élément sert à regrouper les éléments graphiques, Notez l'héritage des propriétés, mais aussi leur redéfinition locale est possible.

Structuration: Eléments de groupages et références

3. Objets abstraits avec <symbol>, <defs> et <use>

- **a. Symboles (élément <symbol>)**
  - Cet élément permet de définir des objets graphiques réutilisables en plusieurs endroits, avec l'élément **<use>**.
  - **<symbol>** ressemble à **<g>**, sauf que l'objet lui-même n'est pas dessiné.
  - Cet élément possède des attributs supplémentaires
- **b. Définitions (élément <defs>)**
  - Cet élément ressemble un peu à **<symbol>**, mais est plus simple. De même, l'objet défini n'est pas dessiné.
- **c. Utilisation d'objets: la balise <use>**
  - **<use>** permet de réutiliser les objets suivants : **<svg>**, **<symbol>**, **<g>**, éléments graphiques et **<use>**.
  - Cet élément se comporte légèrement différemment selon le type d'objet défini. Il s'agit donc d'un instrument de base pour éviter de répéter du code.
  - Les objets réutilisés doivent avoir un identificateur XML. Par exemple, **<rect id="rectanglerouge" fill="red" width="20" height="10"/>**.
  - Les attributs **x**, **y**, **width** et **height** permettent de redéfinir la taille et la position de l'élément appelé.
  - **xlink:href** permet de renvoyer à l'élément défini.

Structuration: Eléments de groupages et références

4. Titre <title> et descriptions <desc>

- Ces éléments permettent de documenter le code.
- Ils ne sont pas affichés tels quels. En revanche, un "client" peut par exemple décider de les afficher comme des infobulles.
- Ces documentations peuvent être utiles pour principalement deux raisons :
  - éventuellement mieux comprendre le code (même s'il est bien sûr toujours possible de le faire *via* des commentaires <!-- (...) --!>),
  - mais aussi, et surtout, permettre un meilleur référencement du SVG par un moteur de recherche.

Insertion d'images : <image>

Les formats supportés sont jpeg et png.

La balise <image> permet également d'insérer un autre fichier SVG.

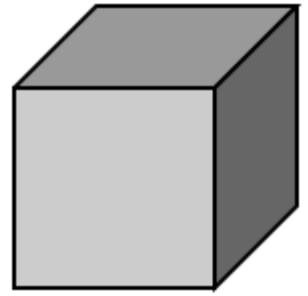
Les attributs sont :

- x et y, qui définissent la position de l'image ;
- width et height, qui définissent la taille de l'image ;
- xlink:href indique l'URI de l'image (équivalent de l'attribut src de la balise <img> en HTML.
- Cet élément possède également un attribut supplémentaire, preserveAspectRatio qui permet de définir la manière dont l'affichage de l'image doit s'adapter à son cadre

Animation de base :<animate>

Création d'un graphique de base:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg SYSTEM "svg10.dtd">
<svg width="500" height="500" xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <symbol id="cube" stroke="black" stroke-line-join="bevel" stroke-width="2">
      <rect width="100" height="100" fill="#ccc" x="1" y="42" />
      <polygon points="1,42 42,1 142,1 101,42 1,42" stroke-width="2" fill="#999" />
      <polygon points="101,42 142,1 142,101 101,142 101,42" fill="#666" />
    </symbol>
  </defs>
  <use xlink:href="#cube" x="150" y="150" />
</svg>
```



Animation de base :<animate>

Animation d'un attribut:

```
<use xlink:href="#cube" x="150" y="150">
  <animate
      attributeName="y"
      dur="2s"
      values="150; 140; 130; 120; 110; 100; 110; 120; 130; 140; 150"
       repeatCount="5"/>
</use>
```

Il est possible de commencer ou finir à des instants déterminés (attributs begin ou end), mais aussi de synchroniser différentes animations.

Animation de base :<animate>

Utilisation de la souris**:**

- Il est facile d'insérer un comportement lié à la souris. Par exemple, begin="mouseover" déclenche l'animation dès que l'on survole l'élément par la souris.
- Nous allons plutôt créer un petit bouton, afin de tester ce genre de comportement.
- Il suffit de créer un rectangle, que l'on identifie par un id="bouton", par exemple, et d'écrire dans notre élément <animate> un begin="bouton.click", pour obtenir l'effet voulu.

Animation de base :<animate>

Utilisation de la souris**:**

```
<rect width="80" height="15" stroke="black" fill="#0ee" id="bouton"/>
<text x="10" y="12" stroke="black" stroke-width="1" id="textbouton">Lancer!</text>
<use xlink:href="#cube" x="150" y="150">
 <animate attributeName="y" dur="2s" values="150; 140; 130; 120; 110; 100; 110; 120; 130; 140; 150" begin="bouton.click"/>
</use>
```

Animation de base :<animate>

Plus de contrôle:

- **a. Attributs from et to**
  - Ces deux attributs permettent de ne pas avoir à spécifier une liste de valeurs
  - On peut par exemple ajouter **<animate attributeName="x" dur="3s" from="150" to="100"/>** à l'animation précédente pour modifier également la position horizontale.

Animation de base :<animate>

b. Figer une animation

- Pour figer l'animation dans son état final, on utilise l'attribut **fill**, avec la valeur **freeze** (geler, en anglais). On obtient ainsi le résultat visible sur l'exemple suivant...
- <animate

  attributeName="x"

  dur="3s"

  from="150"

  to="100"

  fill="freeze"/>

Animation de base :<animate>

C. Répéter une animation

- On peut également demander à ce qu'une animation se répète un nombre déterminé de fois, à l'aide de l'attribut repeatCount.
- Cet attribut peut prendre comme valeur un nombre entier, ou bien indefinite, qui permet de boucler à l'infini.
-

```
<animate
  attributeName="y"
  dur="3s"
  values="150; 140; 130; 120; 110; 100; 110; 120; 130; 140; 150"
  repeatCount="indefinite"/>
```

Animations plus complexes

1. Changements de couleurs : l'élément <animateColor>

- La couleur nécessite un traitement séparé.
- Il est réalisé à l'aide de l'élément <animateColor>, mais les attributs de base restent les mêmes.
- La couleur à modifier doit avoir été fixée lors de l'appel à l'élément, pas lors de la définition
- **<animateColor**
  attributeName="fill"
  attributeType="CSS"
  values="#ccc; #000"
  dur="3s" />

Animations plus complexes

2. Rotations, mises à l'échelle et translations

- La syntaxe de l'attribut transform ne se prête pas à l'utilisation de l'élément <animate>.
- Il a été nécessaire de développer un autre élément, <animateTransform>.
- Il faut renseigner l'attribut attributeName par la valeur transform.
- Cet élément utilise un attribut, type, qui lui permet de déterminer de quel type de transformation il va s'agir.

Animations plus complexes

- **Le code suivant permet de décaler dans la direction des x et des y positifs l'élément à animer:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg SYSTEM "../svg10.dtd">
<svg width="500" height="500" xmlns:xlink="http://www.w3.org/1999/xlink">
   <defs>
         <rect width="100" height="100" id="motif"/>
   </defs>
   <use xlink:href="#motif" x="100" y="100" fill="lime">
         <animateTransform attributeName="transform" type="translate" from="0,0"
   to="100,100" dur="3s"/>
   </use>
</svg>
```

Animations plus complexes

- **Pour la rotation:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg SYSTEM "../svg10.dtd">
<svg width="500" height="500" xmlns:xlink="http://www.w3.org/1999/xlink">
    <defs>
            <rect width="100" height="100" id="motif"/>
    </defs>
    <use xlink:href="#motif" x="100" y="100" fill="lime">
            <animateTransform
                    attributeName="transform"
                    type="rotate"
                    from="0,100,100"
                    to="360,100,100"
                    dur="3s"/>
    </use>
</svg>
```

Animations plus complexes

- **3. Superpositions d'effets**
  - **Par défaut, à chaque fois que l'on ajoute une animation, celle-ci écrase celles qui ont été définies avant.**
  - **Afin d'éviter ce comportement, il faut renseigner l'attribut additive avec la valeur sum (addition).**
  - Exemple:

```
<use xlink:href="#rectangle" x="200" y="200">
  <animate attributeName="x" begin="bouton.click" dur="3s" to="50" repeatCount="indefinite" />
  <animateTransform attributeName="transform" type="rotate" from="0,220,200" to="720,220,200"
  dur="5s" additive="sum" repeatCount="indefinite"/>
  <animateTransform attributeName="transform" type="translate" values="0,0;0,100;0,0" additive="sum"
  dur="5s" repeatCount="indefinite"/>
</use>
```

Exemple de document SVG

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg SYSTEM "svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0">
  <symbol id="logo">
    <rect width="200" height="100" x="10" y="10" stroke="blue" stroke-width="3" fill="yellow"></rect>
   <circle r="50" cx="100" cy="100" stroke="green" fill="red" stroke-width="3" fill-opacity="0.5"></circle>
  </symbol>
   <g id="logo2">
    <use xlink:href="#logo" x="100" y="100"></use>
     <use xlink:href="#logo" x="200" y="200"></use>
     <use xlink:href="#logo" x="100" y="300"></use>
   </g>
   <use xlink:href="#logo2" x="10" y="10">
      <animateTransform attributeName="transform" type="rotate" dur="3s" begin="1s" from="0,100,100"
          to="360,300,300" repeatCount="indefinite"></animateTransform>
   </use>
</svg>
```

XSL- SVG

- On considère un document XML qui présente les mesures des températures des différentes villes à une date donnée.
- Chaque mesure est qualifiée par une date et formée par une ensemble de villes
- Chaque ville est qualifiée par un nom et une température.
- Travail à faire:

1-Faire une représentation graphique de l'arbre XML

2-Créer une DTD

3- Créer un schéma XML

4- Créer le document XML

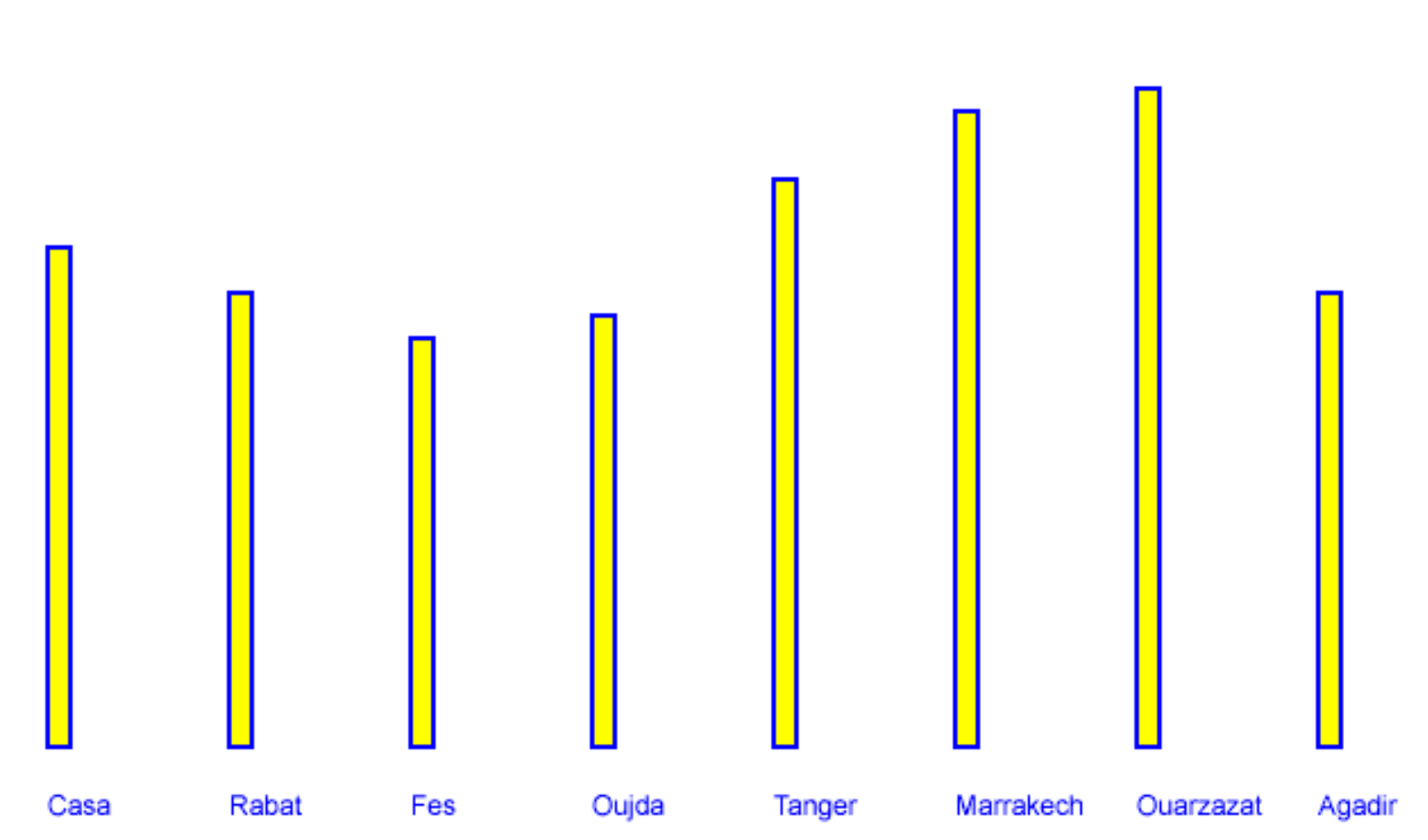
5- Créer une feuille de style qui permet d'afficher une page HTML

6- Créer une feuille de style qui permet de transformer ce document en un document SVG (voir fig.1)

```
<?xml version="1.0" encoding="UTF-8"?>

<meteo>
   <mesure date="2006-1-1">
      <ville nom="Casa" temperature="22"/>
      <ville nom="Rabat" temperature="20"/>
      <ville nom="Fes" temperature="18"/>
      <ville nom="Oujda" temperature="19"/>
      <ville nom="Tanger"  temperature="25"/>
      <ville nom="Marrakech"  temperature="28"/>
      <ville nom="Ouarzazat"  temperature="29"/>
      <ville nom="Agadir"  temperature="20"/>
   </mesure>
   <mesure date="2006-1-2">
      <ville nom="Casa" temperature="21"/>
      <ville nom="Rabat" temperature="23"/>
      <ville nom="Fes" temperature="19"/>
      <ville nom="Oujda" temperature="20"/>
      <ville nom="Tanger"  temperature="23"/>
      <ville nom="Marrakech"  temperature="27"/>
      <ville nom="Ouarzazat"  temperature="25"/>
      <ville nom="Agadir"  temperature="23"/>
   </mesure>
</meteo>
```

Résultat de transformation de la deuxième feuille de style

Températures à la date : 2006-1-1



Feuille de style XSL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml" media-type="image/svg+xml" indent="yes" />
  <xsl:template match="/">
     <svg xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg">
        <line x1="20" y1="20" x2="20" y2="400" stroke="blue" stroke-width="2"></line>
        <line x1="20" y1="400" x2="800" y2="400" stroke="blue" stroke-width="2"></line>
        <xsl:for-each select="meteo/mesure[@date='2006-1-1']">
           <xsl:for-each select="ville">
              <xsl:variable name="temp" select="@temperature"></xsl:variable>
              <xsl:variable name="H" select="$temp*10"></xsl:variable>
              <xsl:variable name="XR" select="position()*80"></xsl:variable>
              <xsl:variable name="YR" select="400-$H"></xsl:variable>
              <text x="{$XR}" y="420" stroke="blue">
                 <xsl:value-of select="@nom"/>
              </text>
              <rect width="10" height="{$H}" x="{$XR}" y="{$YR}" stroke="blue" stroke-width="2"
                fill="yellow">
                 <animate attributeName="height" dur="3s" from="0" to="{$H}"></animate>
                 <animate attributeName="y" dur="3s" from="400" to="{$YR}"></animate>
              </rect>
           </xsl:for-each>
        </xsl:for-each>
     </svg>
  </xsl:template>
</xsl:stylesheet>
```

Suite

```
<xsl:for-each select="meteo/mesure[@date='2006-1-1']">
            <xsl:for-each select="ville">
               <xsl:variable name="temp" select="@temperature"/>
               <text x="{position()*80}" y="430" font-size="12" fill="blue">
                  <xsl:value-of select="@nom"/>
               </text>
               <rect width="10" height="{$temp*10}" stroke="blue" stroke-width="1" fill="yellow"
                x="{position()*80}" y="{400-$temp*10}">
                  <animate attributeName="height" dur="3s" from="0" to="{$temp*10}" fill="freeze"></animate>
                  <animate attributeName="y" dur="3s" from="400" to="{400-$temp*10}" fill="freeze"></animate>
               </rect>
            </xsl:for-each>
         </xsl:for-each>
         </g>
      </svg>
</xsl:template>
</xsl:stylesheet>
```