Les technologies XML

Denis MONASSE
Lycée Louis le Grand

29 avril 2002

1 Introduction à XML

1.1 Qu'est ce que XML

XML (pour *eXtensible Markup Language*) est un ensemble de règles pour la conception de fichiers textes permettant de structurer des données sous forme arborescente. XML facilite la réalisation de fichiers qui ne soient pas ambigus, et qui évitent les pièges courants, tels que la non-extensibilité, l'absence de prise en charge de l'internationalisation/localisation et la dépendance par rapport à certaines plates-formes. Pour cela XML utilise des balises analogues aux balises HTML ou aux environnements de LaTeX ; ces balises sont des mots encadrés par des < et des > et elles sont susceptibles de comporter des attributs sous la forme `nom="valeur"`. Mais alors qu'en HTML ces balises sont prédéfinies et non extensibles et qu'en LaTeX les environnements sont prédéfinies et extensibles par l'utilisateur, les balises XML sont là seulement pour délimiter les éléments de données. Une balise `<p>` peut aussi bien encadrer un paragraphe dans un fichier XML destiné à afficher une page Web qu'encadrer un prix dans un catalogue de vente par correspondances, au choix de son utilisateur.

Les fichiers XML sont des fichiers textes contenant des caractères relevant de différents codages (typiquement ASCII PC, ASCII Mac ou Unicode). Ils sont destinés à contenir des données et non à être lus. L'avantage de stocker les données dans des fichiers textes (par rapport à un stockage dans un format binaire) est de permettre un débogage facile des applications ; de plus un expert peut relativement aisément déceler la structure d'un fichier ou même corriger directement un tel fichier à l'aide d'un éditeur de textes, si le fichier est par exemple légèrement endommagé.

La contrepartie du stockage des données sous forme de texte en utilisant un système de balises est un encombrement important des fichiers XML (analogue à l'encombrement des fichiers TeXou Postscript). C'est un problème mineur à l'heure actuelle alors que les disques durs de grande capacité sont d'usage courant, que les débits sur les réseaux deviennent de plus en plus grands ; d'autre part les fichiers XML (comme les fichiers TeX ou les fichiers Postscript) sont facilement comprimés avec des rapports de compression très intéressants.

L'histoire de XML est relativement récente puisque l'on peut dater sa naissance de 1996. En fait, XML est l'héritier du langage SGML qui est un langage de balisage né au début des années 1980, normalisé en 1986, mais dont la complexité a limité l'usage à des projets de documentation de taille très importante. Un langage intermédiaire a été le langage HTML qui sert à afficher les pages Web depuis 1990. Les concepteurs de SGML ont simplement pris les meilleures parties de SGML, tiré partie de l'expérience de HTML, de ses limitations et de ses défauts, pour définir une technologie aussi puissante que SGML sans en avoir la complexité.

Nous verrons d'autre part que XML est modulaire, indépendant des plate-formes, des encodages et même des alphabets, et, cerise sur le gâteau, qu'en plus d'être une norme reconnue, il est entièrement libre de droits.

1.2 De HTML à XML

HTML est un langage de présentation de documents : les balises servent à structurer en-têtes, titres, paragraphes, images, polices de carcactères, style, etc. Il souffre de graves défauts qui le rendent peu apte à structurer des données :

- HTML n'est pas extensible : le jeu de balises est fixé une fois pour toutes et la seule manière de l'enrichir est de faire reconnaître l'usage de nouvelles balises par le World Wild Web Consortium et les principaux fabricants de butineurs Web, vaste programme
- HTML est uniquement orienté sur la façon dont il faut afficher les données à l'écran, pas du tout vers la structure de ces données

– Le passage d'un logiciel (traitement de textes, tableur, gestion de fichiers) à un fichier HTML permettant la diffusion de données sur le Web peut être automatisé, mais la moindre modification de ces données nécessite de refaire toute la traduction et le reformatage de ces données

– HTML peut présenter des données, mais il ne peut les présenter que d'une seule manière (pas question qu'une même page HTML puisse afficher un tableau trié suivant différents critères au gré de l'utilisateur)

– HTML présente les données par la façon de les afficher, mais il ne donne aucune indication sur leur signification ; autrement dit un fichier HTML contient une syntaxe, mais pas de sémantique : le mot *marche* renverra aussi bien à un fabricant d'escaliers, à un site sur les jeux olympiques ou à un site sur le pasteur Martin Luther King ; c'est là l'un des gros problèmes des moteurs de recherche sur Internet, très partiellement comblé par l'usage des balises META fournissant des listes de mots clés.

Prenons un exemple d'un site personnel contenant quelques recettes maison, voilà un exemple de fichier HTML auquel ce site pourrait faire appel.

```
<!-- Une recette HTML -->
<HTML>
<HEAD>
<TITLE>Le ragout du père Denis</TITLE>
</HEAD>
<BODY>
<H3>Le ragout du père Denis</H3>
Une recette bien de chez nous.
<H4>Ingredients</H4>
<TABLE BORDER="1">
<TR BGCOLOR="#308030"><TH>Qté</TH><TH>Unité</TH><TH>Element</TH></TR>
<TR><TD>1</TD><TD>boîte</TD><TD>camembert</TD></TR>
<TR><TD>800</TD><TD>g</TD><TD>sauté d'agneau</TD></TR>
<TR><TD>500</TD><TD>ml</TD><TD>vin blanc</TD></TR>
<TR><TD>2</TD><TD></TD><TD>marshmallows</TD></TR>
</TABLE>
<P>
<H4>Instructions</H4>
<OL>
<LI>Faire fondre les marshmallows dans le vin blanc...</LI>
<!-- et ainsi de suite -->
</BODY>
</HTML>
```

avec comme affichage (si votre browser savait afficher les caractères accentués)

Les avantages sont évidents : le code est relativement lisible, tout navigateur est capable d'afficher le fichier (même si l'affichage peut dépendre du logiciel), l'affichage peut être facilement modifié à l'aide de feuilles de style en cascade (les fameuses CSS).

Par contre, la signification des différents champs n'est pas évidente : il est difficile de prendre un fichier HTML et d'en extraire des informations pertinentes.

Regardons la forme que pourrait prendre le même fichier en XML :

```
<?xml version="1.0"?>
<Recette>
   <Nom>Le ragout du père Denis</Nom>
   <Description>
     Une recette bien de chez nous.
   </Description>
   <Ingredients>
      <Ingredient>
         <Qté unité="boîte">1</Qté>
         <Element>camembert</Element>
      </Ingredient>
      <Ingredient>
         <Qté unité="g">500</Qté>
         <Element>sauté d'agneau</Element>
      </Ingredient>
      <Ingredient>
         <Qté unité="ml">500</Qté>
         <Element>vin blanc</Element>
      </Ingredient>
      <Ingredient>
         <Qté>3</Qté>
         <Element>marshmallows</Element>
      </Ingredient>
   </Ingredients>
   <Instructions>
      <Etape>
Faire fondre les marshmallows dans le vin blanc...
      </Etape>
      <!-- et ainsi de suite ... -->
   </Instructions>
</Recipe>
```

Le fichier décrit la recette en terme d'ingrédients et d'instructions rangées séquentiellement en étapes. Chacun des ingrédients est décrit par une quantité (dans une certaine unité facultative) de certains éléments. Ici on décrit ce qu'est une recette de cuisine, plutôt que la façon de l'afficher.

1.3 La syntaxe XML

La syntaxe XML est particulièrement simple. Les balises, analogues aux balises d'environnement de LaTeX `\begin{...}` et `\end{...}`, sont copiées sur les balises HTML (ou SGML) sous la forme `<...>` pour une balise de début et `</...>` pour une balise de fin. Comme en LaTeX(et contrairement à HTML), ces balises doivent obligatoirement conduire à des expressions bien formées : toute balise de début doit correspondre à une balise de fin, les balises doivent être correctement imbriquées, aucun recouvrement n'étant possible. Par contre le choix des balises délimitant les éléments est laissé entièrement au bon vouloir de l'utilisateur, à condition que ce soit des identificateurs XML.

Un exemple correct :

```
<document><partie>...</partie><section>...</section></document>
```

et deux exemples incorrects

- `<document><partie>...<section>...</partie></section></document>` (recouvrement)

– <document><partie>...</partie><section>...</document> (il manque une balise de fin)

On appelle **élément XML** tout ce qui est compris entre une balise de début et la balise correspondante de fin (y compris ces deux balises). Un élément a un contenu (tout ce qui est compris au sens strict entre les deux balises) qui peut être un contenu simple (du texte pur), un ou plusieurs éléments, un mélange de texte et d'éléments XML (contenu mixte) ou même un contenu vide. Un raccourci permet d'écrire de manière abrégée un élément dont le contenu est vide sous la forme <.../>. C'est ainsi que l'élément <newline/> est équivalent à l'élément <newline></newline>.

Chaque élément peut en outre disposer d'attributs. Chacun de ces attributs a un nom (un identificateur XML) et un contenu (qui doit être obligatoirement une chaîne de caractère, même si cette chaîne de caractère peut être interprêtée comme un nombre). Au choix, les délimiteurs de chaînes de caractères peuvent être l'apostrophe ou le guillemet (à condition d'utiliser le même délimiteur au début et à la fin de la chaîne de caractères). Ces attributs sont définis à l'intérieur de la balise de début de l'élément XML sous la forme nom="valeur" ou nom='valeur'. Voici un exemple d'élément XML

```
<font name="Times" size ="9" face='bold'>Luminy</font>
```

On soulignera également que la norme XML interdit l'usage des caractères <,>,",' pour tout autre usage que les balises XML. A tout autre endroit ils doivent être remplacés par leurs alias >, <, " et '. Le lecteur se convaincra rapidement de la nécessité d'une telle règle, dans la mesure où le langage XML ne contient (presque) pas de mot réservé.

1.4 Le document XML

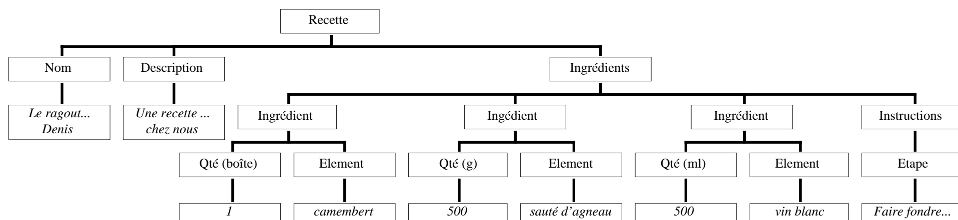
Les règles suivantes définissent un document XML syntaxiquement correct :

1. le document commence par la déclaration <?xml version="1.0"?>
2. il existe un unique élément racine (délimité par une balise d'ouverture et une balise de fermeture) contenant tous les autres éléments
3. tous les élements (délimités par une balise d'ouverture et une balise de fermeture) doivent être proprement imbriqués, sans recouvrement.

De cette manière, chaque document XML correspond de manière bijective a un arbre syntaxique XML : les noeuds (internes) de cet arbre sont étiquetés par les identificateurs des balises du document, chacun de ces noeuds disposant éventuellement d'une liste d'attributs, les feuilles (ou noeuds externes) contiennent du texte pur et elles sont éventuellement vides (on peut également stocker des données binaires à l'aide d'une balise spécifique). Chacun des éléments correspond donc à un sous-arbre de l'arbre syntaxique du document, dont la racine est étiquetée par l'identificateur de la balise de cet élément. Un arbre syntaxique XML peut donc (en première approximation) être décrit par le type Caml

```
type XML_tree =
      |  Feuille of string
     |  Element of string*(string*string list)*(XML_tree list);;
                  (* (balise,attributs,branches) *)
```

Voici l'arbre syntaxique XML correspondant à une partie de notre recette du *ragout du père Denis* :



1.5 Les espaces de noms

Le risque est grand, lorsque plusieurs utilisateurs, travaillant dans des domaines proches, utilisent leurs propres balises XML, qu'ils utilisent les mêmes noms de balises XML avec des structures sous-jacentes différentes. Il est donc intéressant de faire en sorte que les noms de balises deviennent en quelque sorte locaux, soit à un fichier, soit à un utilisateur, soit à un grand domaine d'utilisation. Les espaces de noms correspondent à la notion de `package` en Java (notion que l'on retrouve de manière simplifiée en Caml Light et de manière développée en OCaml). Ce sont des préfixes qui s'ajoutent au début d'un nom et qui déclarent son appartenance à cet espace de nom (ou ce package). Chacun utilise couramment en Caml Light les fonctions `random__int` et `random__float` qui désignent les fonctions `int` et `random` du package `random`. Tout programmeur Java fera appel pour afficher un résultat dans la console à la méthode `java.lang.System.out.println` qui désigne la méthode `println` attachée à l'objet `out` (le fichier de sortie standard) qui est un des champs de l'objet `System` appartenant au package `java.lang`. Il est recommandé à tout programmeur Java qui construit une bibliothèque de l'inclure dans un package dont le nom est construit de manière standard de la manière suivante : si le site Web virtuel correspondant à ma bibliothèque possède l'URL `geom.java.monasse.fr`, le nom du package correspondant sera `fr.monasse.java.geom` et l'emplacement du fichier pourra être dans le répertoire `fr/monasse/java/geom`.

La méthode est similaire pour définir un espace de nom propre à un ou plusieurs utilisateurs, mais ce sera le système de fichier qui sera utilisé la place du `point` du langage de programmation. Autrement dit, un espace de nom sera un URI (pour Uniform Resource Idenfifier). La syntaxe est la même que celle d'un URL du Web. Ainsi un espace de nom pourra être associé à l'URI `http://www.w3schools.com/furniture`. Remarquons que l'URI ne sera pas utilisée en tant qu'URL pour aller rechercher des informations sur un site donné. C'est simplement une convention qui permet de donner un unique identifiant à un espace de noms, c'est là son seul rôle. Comme il n'est pas question de faire précéder chaque nom de balise appartenant à un certain espace de nom d'un identifiant aussi compliqué, on définit un alias de l'espace de nom à l'aide d'un attribut d'espace de nom standardisé

`xmlns:namespace-prefix="namespace"`

soit par exemple

`xmlns:mona="http://www.monasse.com/java/geom"`

A l'intérieur de l'élément XML possédant cet attribut (qui peut être le document XML tout entier), le préfixe `mona` désignera l'URI `http://www.monasse.com/java/geom`. Les noms appartenant à cet espace seront préfixés par cet alias sous la forme `alias:element_name`.

Voici un très court exemple d'utilisation d'espace de noms :

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
<h:tr>
<h:td>Pommes</h:td>
<h:td>Bananes</h:td>
</h:tr>
</h:table>
```

De cette manière, l'utilisateur peut conserver l'usage des balises `table`, `tr`, `td` (ou `mona:table`, `mona:tr`, `mona:td`) avec une toute autre signification que les standards HTML, tout en utilisant les balises standard HTML.

1.6 Les DTD

La contrepartie de la liberté totale de définir la structure d'un document XML est bien entendu l'anarchie qui peut en résulter. Chaque utilisateur peut définir lui-même sa structure de document, ses propres balises, ce qui interdit toute normalisation de format de fichier, et donc toute possibilité d'échanger ces fichiers entre deux utilisateurs. Pour remédier à cet état de fait, il faut définir des modèles de fichier. C'est à cela que servent les DTD, c'est à dire les Définitions de Types de Documents (Document Type Definition). Avec ces DTD, XML devient un métalangage capable de décrire des langages spécialisés.

Une DTD va décrire quelle doit être la racine d'un fichier XML et pour chaque balise ou élément quels sont les attributs autorisés et/ou indispensables, quels sont les fils autorisés et/ou indispensables avec leur ordre, ainsi que les types de données qu'ils peuvent éventuellement contenir. Avec une DTD

- chaque document peut inclure une description de sa structure
- plusieurs utilisateurs peuvent se mettre d'accord sur un format de fichier leur permettant d'échanger des données

– une application ou un utilisateur peut vérifier que des données sont valides

La définition d'un type de document se fait sous la forme d'une *instruction* XML de la forme

```
<!DOCTYPE root-element [element-declarations]>
```

L'identificateur `root-element` décrit le type de la racine du document. Les déclarations d'éléments vont décrire d'une part la structure même de l'arbre d'autre part les attributs de ces éléments. Commençons par un petit exemple commenté :

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

Cette DTD décrit un document de type `note`. Celui-ci doit contenir un élément de balise `note` qui lui même doit contenir successivement et dans cet ordre un élément de balise `to`, un élément de balise `from`, un élément de balise `heading` et un élément de type `body`. Chacun des éléments de type `to`, `from`, `heading` et `body` doit contenir uniquement des données de type caractère.

La déclaration d'un élément se fait sous l'une des deux formes

```
<!ELEMENT element-name category>
```

ou

```
<!ELEMENT element-name (element-content)>.
```

Les catégories possibles sont `EMPTY` (vide) ou `ANY` (n'importe quoi). Les contenus sont décrits par un langage inspirés des expressions rationnelles mélant des listes d'éléments séparés par des virgules (qui décrivent des fils devant apparaître dans cet ordre), le symbole de réunion `|` qui décrit une alternative, les symboles de répétition `*` (zéro,une ou plusieurs fois), `+` (une ou plusieurs fois), `?` (zéro ou une fois), `#PCDATA` qui décrit des données de type caractères. C'est ainsi qu'une déclaration d'élément de la forme

```
<!ELEMENT note (#PCDATA|to+|from|(header,message))*>
```

décrira un élément de balise `note` contenant soit une donnée de type caractère, soit un ou plusieurs éléments de balise `to`, soit un élément de balise `from`, soit la succession d'un élément de balise `header` et d'un élément de type `message`, le tout répété zéro, une ou plusieurs fois.

Pour tout élément déclaré, on pourra spécifier chaque attribut sous la forme

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

où `element-name` désigne le nom (la balise) de l'élément, `attribute-name` le nom de l'attribut, `attribute-type` le type de l'attribut et `default-value` sa valeur par défaut. Le type de l'attribut sera la plupart du temps `CDATA` ce qui désigne une chaîne de caractère. La valeur par défaut sera soit une valeur qu'il faut associer à l'attribut si celui-ci n'est pas défini par l'utilisateur, soit `#IMPLIED` qui signifie que l'attribut est facultatif, soit `#REQUIRED` ce qui signifie que l'utilisateur doit donner un valeur à cet attribut pour que la définition de type de document soit respectée.

La combinaison des spécifications de la structure et des attributs permet de définir le type de document. La DTD peut être interne au document (comme on l'a vu ci dessus) mais elle peut aussi être contenue dans un fichier texte externe auquel on fait référence dans le document XML sous la forme

```
<!DOCTYPE root-element SYSTEM "filename">
```

où `filename` désigne le fichier contenant la DTD (il porte par convention un suffixe `.dtd`) sous la forme d'une URI (analogue d'une URL sur le Web).

1.7 Analyseurs validants et non validants

Les analyseurs XML peuvent être de deux types : validants ou non validants. Un analyseur validant vérifiera la conformité du fichier à la définition de type de document (la DTD) qu'il contient ou à laquelle il fait référence (au besoin en allant chercher la DTD sur le Web). Un analyseur non validant se contentera d'analyser la structure du fichier, sans vérifier sa conformité à la DTD.

2 Les analyseurs XML

2.1 Les API

Une interface de programmation d'application (Application Programming Interface, en abrégé API) est un moyen standardisé d'accéder à certaines fonctionnalités, indépendant du langage ou de la façon dont les structures de données sont implémentées. Les analyseurs XML sont définis par le W3C (World Wide Web Consortium) sous forme d'API. Le programmeur doit donc éviter d'accéder directement aux structures de données (que ce soit les fichiers XML ou les arbres XML) et n'utiliser que des *méthodes* standardisées de manipulation des *objets* relatifs à la technologie XML.

Nous classerons les analyseurs syntaxiques XML suivant deux catégories définies par deux API standardisées par le W3C : les analyseurs événementiels implémentant les fonctionnalités de l'API SAX (Simple API for XML) et les analyseurs arborescents implémentant les fonctionnalités de l'API DOM (Document Object Model). Pour préciser ces API, nous utiliserons le langage Java pour lequel on dispose de multiples implémentations de ces deux types d'analyseurs. Les packages Java les plus riches (comme le package Xerces du projet Apache soutenu par Sun et IBM) implémentent les deux API SAX et DOM en construisant de surcroît des analyseurs validants, qui vérifient la conformité d'un document à sa DTD (Définition de Type de Document).

2.2 Analyseurs événementiels : SAX

Le but d'un analyseur événementiel est de lire un fichier XML et de déclencher des évènements standardisés à chaque fois qu'il rencontre un élément de la structure XML. Autrement dit, l'arbre XML n'est pas chargé en mémoire mais tout se passe comme si on assistait à un parcours d'Euler de l'arbre XML avec pour chaque noeud une action préfixe (déclenchement d'un évènement `start...`) et une action postfixe (déclenchement d'un évènement `end...`). Le fait que l'arbre ne soit pas chargé en mémoire permet à cette API de traiter des documents XML de très grande taille (de grandes banques de données ou des ouvrages entiers) indépendamment des capacités de l'ordinateur que ce soit en mémoire ou en capacités de communication.

L'API standard qui définit l'interface d'un analyseur évènementiel pour XML s'appelle SAX (pour Simple API for XML). Voici une interface simplifiée en Java d'une implémentation de SAX :

```
/**
  * Receive notification of the beginning of a document.
  *
  */
public abstract void startDocument ()
  throws SAXException;


/**
  * Receive notification of the end of a document.
  *
  */
public abstract void endDocument ()
  throws SAXException;


/**
  * Receive notification of the beginning of an element.
```

```
    *
    * <p>The Parser will invoke this method at the beginning of every
    * element in the XML document.</p>
    *
    */
  public abstract void startElement (String name, AttributeList atts)
    throws SAXException;


  /**
    * Receive notification of the end of an element.
    *
    * <p>The SAX parser will invoke this method at the end of every
    * element in the XML document.</p>
    *
    */
  public abstract void endElement (String name)
    throws SAXException;


  /**
    * Receive notification of character data.
    *
    * <p>The Parser will call this method to report each chunk of
    * character data.  SAX parsers may return all contiguous character
    * data in a single chunk, or they may split it into several
    * chunks.</p>
    *
    */
  public abstract void characters (char ch[], int start, int length)
    throws SAXException;
```

On voit donc que l'analyseur gère 5 types d'évènements : le début d'un document, la fin d'un document, le début d'un élément (en transmettant en paramètres le nom de l'élément et une liste d'attributs), la fin d'un élément (en transmettant en paramètre le nom de l'élément), la rencontre de données de type caractères (en transmettant un tableau des caractères rencontrés).

En fait l'API SAX comprend également des méthodes standard pour créer un analyseur, pour lancer cet analyseur sur un document XML considéré lui-même comme un objet. Il reste au programmeur à gérer les évènements transmis par l'analyseur pour traiter le fichier XML, en concrétisant les méthodes abstraites décrites dans l'interface ci-dessus. Pour cela de nombreux outils standardisés sont fournis dans les packages implémentant l'API SAX.

Remarquons que les packages Java les plus petits implémentant les fonctionnalités indispensables de SAX ne font que 7 kilooctets (l'un d'entre eux se nomme MiniXML, il suffit amplement au traitement de petits fichiers XML comme ceux que peuvent créer des logiciels simples) ce qui les rend particulièrement adaptés à l'utilisateur lambda qui désire par exemple écrire des Applets utilisant les fonctionnalités de XML. Bien entendu ces packages réduits construisent des analyseurs non validants, c'est à dire qui ne vérifient pas la conformité du document à un DTD (Document Type Definition).

2.3 Analyseurs arborescents : DOM

L'analyse évènementielle a bien évidemment ses limites, ne serait-ce que le fait qu'elle limite les possibilités de retour en arrière. Si l'on sait que l'on peut sans grande difficulté reconstituer un arbre à l'aide d'un parcours d'Euler préfixe-postfixe, il peut être agréable d'avoir accès de manière directe et standardisée à une implémentation de l'arbre d'un document XML. C'est là le rôle de l'API DOM (pour Document Object Model).

Le rôle de cette API sera donc de permettre au programmeur de lire un fichier XML, de construire en mémoire l'arbre syntaxique de ce fichier, d'accéder de manière standardisée à la structure de l'arbre et à son contenu (indépendamment

de son implémentation), de manipuler cet arbre et éventuellement de réécrire l'arbre modifié dans un fichier XML.

Le type d'objet standard de cette API est le noeud, sous forme de la classe `Node`. Celle ci dispose d'un grand nombre de méthodes permettant d'avoir accès à la structure de l'arbre syntaxique et éventuellement de la modifier. En voici quelques exemples :

```
Node appendChild(Node newChild)
     // Adds the node newChild to the end of the list of children of this node.
NodeList getChildNodes()
     // A NodeList that contains all children of this node.
Node getNextSibling()
     //  The node immediately following this node.
java.lang.String getNodeName()
     //  The name of this node, depending on its type; see the table above.
short getNodeType()
     //  A code representing the type of the underlying object, as defined above.
java.lang.String getNodeValue()
     //  Returns the value of this node, depending on its type; see the table above.
Document getOwnerDocument()
     //  The Document object associated with this node.
Node getParentNode()
     // The parent of this node.
Node getPreviousSibling()
     // The node immediately preceding this node.
boolean hasAttributes()
     //  Returns whether this node (if it is an element) has any attributes.
boolean hasChildNodes()
     //  Returns whether this node has any children.
Node removeChild(Node oldChild)
     // Removes the child node indicated by oldChild from the list of
     // children, and returns it.
Node replaceChild(Node newChild, Node oldChild)
     //  Replaces the child node oldChild with newChild in the list of children,
     //  and returns the oldChild node.
void setNodeValue(java.lang.String nodeValue)
     //   Sets the value of this node, depending on its type; see the table above.
```

L'objet `Document` hérite lui-même du type `Node` et sert de racine à l'arbre. C'est lui qui est retourné par l'analyseur implémentant l'API DOM.

Remarquons que dans la mesure du possible, l'API SAX (ou son extension JAXP) doit être préférée à l'API DOM. Cette dernière est en effet très gourmande en mémoire et en temps machine (elle a en particulier à déclencher très souvent la mise en route du Garbage Collector pour récupérer la mémoire occupée par tous les petits objets intermédiaires qu'elle a tendance à créer).

3 Le (re)formatage des fichiers XML : CSS et XSL

3.1 La notion de formatage

Une contrepartie de la liberté laissée à l'utilisateur (ou à l'auteur de la Document Type Definition) de définir son propre format de fichier XML est l'impossibilité pour un navigateur d'afficher de manière structurée un tel fichier, malgré la ressemblance superficielle entre XML et HTML. Une autre contrepartie relève de l'incommunicabilité entre les êtres ; deux utilisateurs n'utilisant pas le même format de fichier XML (ou n'utilisant pas la même DTD) sont dans l'impossibilité d'échanger leurs données.

Il s'est avéré rapidement après la définition de XML qu'il était nécessaire de disposer d'outils standardisés pour formater ou reformater les fichiers XML. Déjà à l'époque, on avait vu naître les feuilles de style pour les documents HTML qui permettaient à l'auteur d'un site ou à son utilisateur de définir ses propres styles d'affichage des fichiers

HTML à l'aide de fichiers textes externes (ou internes) facilement modifiables et répondant à une syntaxe simple. Il était naturel d'adapter ces feuilles de style aux fichiers XML pour obtenir un formatage rudimentaire mais efficace de ces fichiers pour affichage dans un navigateur.

Mais l'usage de feuilles de style ne remplissait son rôle d'affichage que de manière rudimentaire et ne permettait pas un reformatage des fichiers XML autorisant un partage de données entre deux utilisateurs utilisant des formats ou des DTD différents. De là est venu le développement d'un véritable langage de transformation de fichiers XML, appelé improprement eXtensible Styling Language (XSL). Ce langage de style extensible est en fait un langage (écrit en XML) qui permet une véritable transformation de l'arbre d'un document, et en particulier la transformation d'un document XML en document HTML pour affichage dans un navigateur.

3.2 Les feuilles de style en cascade : CSS

Une feuille de style en cascade est un document externe (ou interne) à un fichier XML (ou HTML) qui décrit, pour chaque balise, la manière dont le contenu d'un élément XML encadré par cette balise doit être affiché par le navigateur : fond d'écran, police de caractère, taille, couleur et style des caractères. Ces feuilles de style sont dites en cascade car le style appliqué à un élément XML (encadré par une balise) s'applique par défaut en cascade à tous les descendants de cet élément, à moins que ceux-ci ne disposent de leur propre style d'affichage.

Une feuille de style en cascade (en anglais Cascading Style Sheet, ou CSS) est un fichier texte (qui n'est pas un fichier XML, premier reproche de cohérence) qui contient une suite d'instructions du type :

```
balise {propriete: valeur; propriete: valeure; ... }
```

Les balises sont des balises XML (ou HTML) ; les propriétés sont du type `color`, `font-family`, `font-size`, `text-align`, `background-color`. Leurs valeurs peuvent être des mots prédéfinis comme `left`, `right`, `center`, `justify` pour la propriété `text-align`, des noms de police pour la propriété `font-family`, ou des nombres. Les valeurs doivent être mises entre guillemets s'il s'agit de mots composés.

Voici un exemple de feuille de style en cascade :

```
CATALOG
{
background-color: #ffffff;
width: 100%;
}
CD
{
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE
{
color: #FF0000;
font-size: 20pt;
}
ARTIST
{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
Display: block;
color: #000000;
margin-left: 20pt;
}
```

permettant d'afficher le fichier XML suivant

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited with XML Spy v4.2 -->
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
</CATALOG>
```

sous la forme suivante dans n'importe quel navigateur supportant les feuilles de style :

La référence à une feuille de style depuis un fichier XML se fait sous la forme

```
<?xml-stylesheet type="text/css" href=file_name?>
```

où `file_name` désigne le nom du fichier (ou son URI), qui porte par convention le suffixe `.css`.

3.3 Le langage de style et de format XSL

Les feuilles de style en cascade ne donnent que très peu de contrôle sur le fichier XML. Leur seule utilité est de permettre à un navigateur d'effectuer un formatage visuel de ce fichier en vue de l'affichage à l'écran sous une forme assez rudimentaire. Le langage XSL (eXtensible Styling Language ou langage de style extensible) est au contraire un projet ambitieux permettant de transformer un fichier XML possédant un certain format en un autre fichier XML possédant un autre format (et en particulier en un fichier HTML complexe si l'on désire un affichage dans un navigateur). Le fichier peut soit être transformé à la volée par le navigateur qui se chargera ensuite de l'affichage, soit être enregistré dans on nouveau format, permettant à deux utilisateurs de partager des données, même si celles-ci utilisent des formats XML différents (par exemple par l'intermédiaire de DTD différentes).

Le langage de style XSL est un sous-ensemble de XML, dont les noms appartiennent à l'espace de noms

`http://www.w3.org/1999/XSL/Transform`

qu'il est d'usage de référencer à l'aide de l'alias `xsl` si bien que toutes les balises XSL seront du type `xsl:...`. Une feuille de style XSL commencera par le prologue

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

et devra se terminer par `</xsl:stylesheet>`

Le langage XSL procède par reconnaissance de motifs. Le processeur XSL explore récursivement les noeuds de l'arbre syntaxique du fichier XML à partir de la racine de cet arbre (qui est par convention le fichier tout entier, et non la première balise). Pour chaque noeud rencontré, il recherche un motif qui s'applique à ce noeud ; si plusieurs motifs s'appliquent à ce noeud, il choisit le plus spécifique et s'il en reste encore plusieurs, il choisit le dernier. Une fois choisi le motif qui s'applique à ce noeud, il va instancier le modèle (*template* en anglais) associé à ce motif : ce modèle comprendra de la source XML (appartenant à un autre espace de nom que celui référencé par `xsl`), du texte et des instructions XSL qui recopieront (éventuellement en les modifiant) des données provenant du document XML de base.

A tout instant, le processeur XSL dispose d'une part d'un noeud XML courant (analogue au répertoire courant) qui est référencé par le motif "`.`" (le point du répertoire courant sous DOS ou Unix) et d'une liste de noeuds en attente qui seront traité une fois que le traitement du noeud courant sera exécuté.

Les motifs utilisent un langage spécifique appelé XPATH qui décrit les noeuds de l'arbre XML par analogie à un système de fichier. C'est ainsi que l'on a les exemples suivants :

Motif	Noeuds reconnus
`/`	La racine du document
`/toto`	tout noeud de nom `toto` qui est fils de la racine du document
`/toto/*`	tout noeud qui est fils d'un noeud de nom `toto` lui même fils de la racine du document
`/toto//bidule`	tout noeud de nom bidule qui est descendant
	d'un noeud de nom `toto` lui même fils de la racine du document
`.`	le noeud courant
`..`	le père du noeud courant
`./toto`	tout noeud de nom toto qui est fils du noeud courant
`toto`	tout noeud de nom toto qui est fils du noeud courant
`.//toto`	tout descendant du noeud courant de nom `toto`
`toto/@font`	tout attribut de nom `font` d'un noeud de nom `toto`

On constate sur ce dernier exemple que les attributs sont considérés comme des noeuds XML, dont le nom doit être précédé par @.

En plus, les motifs peuvent utiliser l'opérateur logique de réunion `|` et des fonctions booléennes spécifiques (toujours entre guillements) comme la fonction `node()` qui reconnaît tous les noeuds (que ce soient des éléments ou des noeuds textuels, alors que `*` ne reconnaît que les noeuds correspondant à des éléments XML associés à des balises), la fonction `text()` qui reconnaît les noeuds terminaux contenant des données textuelles, `comment()` qui reconnaît les commentaires (en l'absence de tel motif, les commentaires sont ignorés par XSL). Nous ne nous étendrons pas plus sur le langage XPATH dont nous n'avons donné qu'une petite idée. Ce langage comprend également des filtrages avec surveillants (analogue au `match ... with ... when ...` de Caml) ainsi que des fonctions pour effectuer des conversions de textes en nombres ou de nombres en textes, pour effectuer des comptages ou pour traiter des chaînes de caractères.

La reconnaissance de motif se fait à l'aide de l'instruction `xsl:template ...` sous la forme

```
<xsl:template match="motif à reconnaître"> modèle à instancier </template>
```

Le modèle à instancier peut comprendre de la source XML, mélange de données textuelles et de balises XML n'appartenant pas à l'espace de noms défini par l'alias `xsl`, et des instructions XSL dont les plus importantes sont

instruction	sémantique
`apply_templates`	ajoute les fils du noeud courant dans la liste des noeuds à traiter
`apply_templates select="motif"`	ajoute les noeuds reconnus par le motif dans la liste des noeuds à traiter
`value_of`	recopie la valeur du noeud (voir ci dessous)
`value_of select="motif"`	recopie la valeur du (premier) noeud reconnu par le motif (voir ci dessous)
`copy`	recopie le noeud lui même sans ses attributs et sans ses fils
`copy_of`	recopie toute la branche issue du noeud
`copy_of  select="motif"`	recopie tous noeuds reconnus par le motif

La valeur d'un arbre est définie récursivement par : la valeur d'un arbre est égale à la réunion des valeurs de ses branches, la valeur d'un noeud (terminal) textuel est son contenu, la valeur d'un autre noeud terminal est vide.

Le langage XSL contient de plus

(i) une instruction répétitive (en plus de l'instruction récursive `apply_templates`) dont la syntaxe est

```
<xsl:for_each select="motif"> modèle à instancier </xsl:for_each>
```

qui instanciera le modèle pour tous les noeuds reconnus par le motif,

(ii) une instruction conditionnelle

```
<xsl:if test="test booléen"> modèle à instancier </xsl:if>
```

(iii) une instruction de choix (analogue au `switch` de Java ou de C)

```
<xsl:choose>
   <xsl:when test="booléen"> modèle à instancier </xsl:when>
   <xsl:when test="booléen"> modèle à instancier </xsl:when>
   ....
   <xsl:otherwise> modèle à instancier </xsl:otherwise>
</xsl:choose>
```

(iv) une instruction de tri `sort` qui peut suivre immédiatement toute instruction `apply_templates` ou `for_each` et qui effectue les opérations en triant suivant les valeurs de noeuds reconnus par un motif avec la syntaxe

```
<xsl:sort select="motif"\>
```

Prenons un exemple simple : un fichier de noms d'animaux qui peut avoir la structure suivante :

```
<animaux>
   <chien> Medor </chien>
   <chat> Mandarine </chat>
   <chien> Brutus </chien>
   <chat> Clementine> </chat>
   <oiseau> Titi </oiseau>
   <chat> Prunelle </chat>
<animaux>
```

Soumettons le à un processeur XSL avec le fichier XSL suivant

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="animaux">
    <AnimauxTries><chats>
    <xsl:for-each select="chat">
    <xsl:sort select="."/>
        <xsl:value-of select="."/>
    </xsl:for-each>
    </chats>
    <chiens>
    <xsl:for-each select="chien">
    <xsl:sort select="."/>
        <xsl:value-of select="."/>
    </xsl:for-each>
    </chiens>
    <oiseaux>
    <xsl:for-each select="oiseau">
    <xsl:sort select="."/>
        <xsl:value-of select="."/>
    </xsl:for-each>
    </oiseaux>
    </AnimauxTries>
</xsl:template>
```

```
</xsl:stylesheet>
```

On obtiendra le nouveau fichier XML suivant, où les animaux sont classés par race, leurs noms étant triés par ordre alphabétique

```
<?xml version="1.0" encoding="UTF-8"?>
<AnimauxTries>
  <chats> Clementine  Mandarine  Prunelle </chats>
  <chiens> Brutus  Medor </chiens>
  <oiseaux> Titi </oiseaux>
</AnimauxTries>
```

Que s'est-il passé ? Le processeur XSL a lancé sa transformation sur l'arbre XML du fichiers des animaux avec comme première instruction l'instruction implicite (qui existe dans tout processeur normal et standard, c'est à dire non-Microsoft)

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

qui dit que pour tout noeud et pour la racine, l'instruction la plus générale est d'ajouter à la liste des noeuds à traiter tous les fils (cette instruction est masquée dès qu'un motif plus spécifique s'applique à un noeud). Ici la racine a un seul fils de nom `animaux`. Le processeur dépile alors le noeud suivant à traiter qui est le noeud `animaux`, il trouve un motif qui le reconnaît dans le `match="animaux"`. Il instancie donc le modèle, recopie le code XML `<AnimauxTries> <chats>`, puis rencontre l'instruction `for-each select="chat"` qui lui dit de prendre un par un les fils de type `chat`, en les triant suivant leur nom, de recopier leur valeur, puis de recopier le code XML `</chats> <chiens>`, de prendre un par un les fils de type `chien`, en les triant suivant leur nom et ainsi de suite.

On réalisera rapidement que l'on peut effectuer beaucoup de transformations à l'aide de ce langage XSL (et du langage XPATH) dont nous n'avons donné ici qu'une idée rudimentaire et simpliste ; en particulier on peut transformer un fichier XML en fichier HTML (ou transformer un fichier MathML en $\TeX$). Les transformations qui ne peuvent être obtenues à partir de XSL, peuvent toujours l'être à l'aide d'une implémentation de SAX ou de DOM dans un langage de programmation. En conclusion, XSL est une composante essentielle de la technologie XML.

4 Quelques sous-ensembles XML

4.1 XHTML

Le langage XHTML est destiné à devenir le successeur de HTML. C'est un sous-ensemble de XML, largement compatible avec HTML et qui remédie aux principaux défauts de ce dernier auquel il apporte la rigueur et la clarté. XHTML combine tous les éléments de HTML 4/01 avec la syntaxe de XML.

On est arrivé à un point où beaucoup de pages Web, profitant du laxisme des principaux navigateurs, contiennent du code HTML erronné du type

```
<html>
<head>
<title>This is bad HTML</title>
<body>
<h1>Bad HTML
</body>
```

(il manque la balise fermante `</h1>`) ou encore

```
<b><i>This text is bold and italic</b></i>
```

dans lequel les balises sont mal imbriquées.

Ce type de code peut très bien fonctionner "correctement" sur un navigateur donné ou sous un système d'explotation donné, et donner des résultats aberrants (voir ne pas s'afficher du tout) avec un autre navigateur ou un autre type de machines. D'autres *tolérances* de HTML conduisent à des erreurs difficiles à déceler comme le fait que les attributs peuvent ne pas recevoir de valeur ou que ces valeurs puissent être ou non encadrées par des guillemets.

Les règles pour passer de HTML à XHTML sont relativement simples :

1. les éléments (délimités par les balises) doivent être proprement imbriqués
2. tous les éléments XHTML doivent être contenus dans un seul élément racine encadré par les balises `<html>` et `</html>`
3. tous les documents HTML doivent avoir une déclaration DOCTYPE
4. les balises `html`, `head` et `body` doivent être présentes et le titre du document doit être présent à l'intérieur de l'élément `head`
5. les noms de toutes les balises et de tous les attributs doivent être en minuscules (HTML n'est pas case-sensitive, XHTM l'est)
6. tout attribut doit avoir une valeur : le code `<option selected>` est toléré en HTML, mais il doit être remplacé par `<option selected="selected">` en XHTML
7. les valeurs des attributs doivent être impérativement encadrées par des guillemets ou des apostrophes (y compris les nombres)
8. tous les éléments doivent être fermés, y compris les éléments vides comme la balise de passage à la ligne : il faut écrire `<br/>` ou `<br></br>` et pas `<br>`

Voici un modèle de document XHTML minimum :

```
<!DOCTYPE Doctype goes here>
<html>
<head>
<title>Title goes here</title>
</head>
<body>
Body text goes here
</body>
</html>
```

La déclaration DOCTYPE ne fait pas partie du document XHTML lui même ; il n'y a pas de balise fermante. Elle décrit la DTD (définition de type de document) à laquelle se conforme le document XHTML. Il existe à l'heure actuelle trois DTD courantes pour les documents XHTML : la DTD stricte qui n'autorise le document à faire appel à aucune des fonctionnalités de présentation de HTML (un tel document doit impérativement faire appel à une feuille de style en cascade CSS pour pouvoir s'afficher), la DTD de transition qui autorise à utiliser toutes les fonctionnalités de HTML compatibles avec les règles édictées ci-dessus, la DTD de frames qui autorise à utiliser des cadres pour pouvoir afficher plusieurs documents dans une fenêtre.

Puisqu'un document XHTML répond à un DTD, il peut être validé par n'importe quel analyseur XML validant, ce qui garantit qu'il est exempt de faute de syntaxe. Comme c'est également un document XML, il peut être transformé par un processeur XSL en un autre document XML et inversement, un document XML peut être transformé, soit à la volée, soit par création de fichier, en un document XHTML en vue de l'affichage dans un navigateur Web.

4.2 SVG

Le langage Scalable Vector Graphics (graphiques vectoriels redimensionnables) est un langage permettant de décrire des graphiques à deux dimensions en XML. Il dispose de trois types d'objets graphiques :

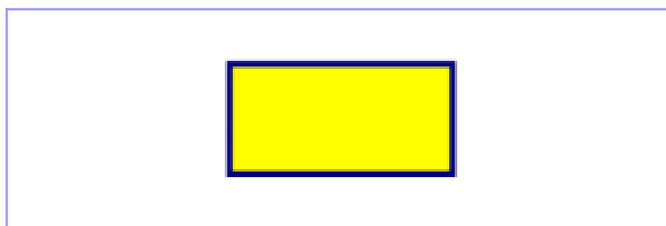
- les formes vectorielles qui sont composées de chemins (comme en Postscript) qui regroupent des segments et des courbes (segments elliptiques, Bezier quadratiques ou cubiques) ; ces formes peuvent être tracées, remplies ou utilisées pour effectuer un clipping (régionnement de l'espace)
- les images bitmap
- le texte

Les objets graphiques peuvent être groupés, transformés, placés dans d'autres objets, on peut leur appliquer des styles. SVG sait gérer le clipping, les couches alpha (effets de transparence et antialisaing), les filtres et les transformations imbriquées.

Voici un exemple de graphique SVG :

```
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
<desc>Example rect01 - rectangle with sharp corners</desc>
<!-- Show outline of canvas using 'rect' element -->
<rect x="1" y="1" width="1198" height="398"
     fill="none" stroke="blue" stroke-width="2"/>
<rect x="400" y="100" width="400" height="200"
     fill="yellow" stroke="navy" stroke-width="10" />
</svg>
```

La balise svg définit le système de coordonnées : la largeur de la fenêtre est fixée à 12 cm, sa hauteur à 4 cm et on utilise un système de coordonnées qui étend l'abscisse de 0 à 1200 et l'ordonnée (ves le bas) de 0 à 400. Ensuite on trace deux rectangles : un rectangle bleu non rempli du point (1,1) au point (1199,399) et un rectangle au bord *navy* rempli de jaune de sommet (400,100) de largeur 400 et de hauteur 200. On obtient le dessin suivant :



Voici un autre exemple qui montre l'utilisation de transformations (ici une translation et une rotation), ainsi que de rectangles arrondis :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
<desc>Example rect02 - rounded rectangles</desc>
<!-- Show outline of canvas using 'rect' element -->
<rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2"/>
<rect x="100" y="100" width="400" height="200" rx="50" fill="green" />
<g transform="translate(700 210) rotate(-30)">
<rect x="0" y="0" width="400" height="200" rx="50"
        fill="none" stroke="purple" stroke-width="30" />
</g>
</svg>
```

La commande de base est, comme en Postscript, la commande de tracé d'un chemin. Sa syntaxe est

```
<path d="definition du chemin"/>
```

dans laquelle la valeur de l'attribut `d` est une chaîne de caractères comprenant des commandes définies par un seul caractère, en minuscule si l'origine de la commande est relative au point courant, en majuscule si les coordonnées sont absolues, suivies des coordonnées numériques des points paramètres de la commande. Les principales commandes relatives sont

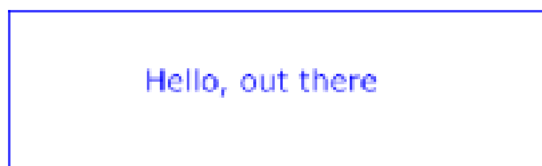
- `m x1 y1` : déplacer de (x1,y1)
- `l x1 y1` : tracer un segment de coordonnées (x1,y1)
- `h x` : tracer un segment horizontal de longueur x
- `v y` : tracer un segment vertical de longueur y
- `c x1 y1 x2 y2 x3 y3 ...` : trace une courbe de Bezier cubique'
- `z` : ferme le chemin

et des analogues absolues en majuscule

On peut insérer du texte dans un dessin :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
    xmlns="http://www.w3.org/2000/svg">
<desc>Example text01 - 'Hello, out there' in blue</desc>
<text x="250" y="150" font-family="Verdana" font-size="55" fill="blue" >
Hello, out there
</text>
<!-- Show outline of canvas using 'rect' element -->
<rect x="1" y="1" width="998" height="298" fill="none" stroke="blue" stroke-width="2" />
</svg>
```

avec comme résultat



Comme en Postscript, on peut définir des transformations affines par une matrice 3×3 du type $\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$ ou en composant des transformations élémentaires du type translation, affinités, homothéties, rotations. Enfin SVG permet d'insérer des images, d'appliquer des filtres de convolution, de créer des animations et même de gérer l'interactivité par l'intermédiaire d'évènements souris ou clavier.

Il s'agit donc d'un projet ambitieux pour créer un format universel, indépendant de la plate-forme pour des fichiers graphiques vectoriels, analogue à Postscript, mais avec une lisibilité et une interéchangeabilité bien supérieure. Remarquons qu'il existe déjà des "scripts" XSL permettant de transformer des fichiers SVG en fichiers PDF.

4.3 MathML

Notre dernier centre d'intérêt sera le langage mathématique MathML. Il s'agit ici d'un langage de syntaxe XML permettant (comme le fait TEX) de décrire la structure d'un texte mathématiques et de ses formules et d'en permettre l'affichage par un navigateur Web (ce que ne permet pas TEX), mais qui permet aussi de décrire un contenu mathématique effectif comme le ferait Mathematica ou Maple. Ici encore, l'avantage par rapport à une source TEX ou LATEX, est d'une part d'avoir une syntaxe identique à celle utilisée par les navigateurs, et de profiter des avantages des feuilles de style CSS ou de préférence XSL pour formater le texte à l'écran ou pour le transformer en d'autres formats XML, mais aussi de pouvoir servir de format natif de fichier pour un outil de calcul formel.

Un texte MathML bien écrit peut être évalué dans Maple, visualisé sur le Web, édité dans un traitement de textes et imprimé sur une imprimante Laser. Le format MathML est soutenu par IBM, Wolfram Research (éditeur de Mathematica), Waterloo Software (éditeur de Maple), l'American Mathematical Society qui développe un traducteur LaTeX-MathML.

MathML admet deux styles d'éléments. Un élément MathML peut-être un simple moyen d'afficher à l'écran une formule mathématique (comme c'est le cas d'une formule contenue dans un document Word, contruite à l'aide de l'éditeur d'équation), mais elle peut aussi décrire la signification mathématique de la formule. C'est ainsi que la formule

$$(a+b)^2$$

peut être codée de deux façons en MathML. Soit avec la présentation typographique

```
<msup>
  <mfenced>
    <mrow>
      <mi>a</mi>
      <mo>+</mo>
      <mi>b</mi>
    </mrow>
  </mfenced>
  <mn>2</mn>
</msup>
```

qui dit que l'on doit avoir une expression avec un exposant, que la base est délimitée (*fenced*) par des parenthèses, qu'elle comprend deux identificateurs a et b et un opérateur $+$, que l'exposant est un nombre. Mais on peut aussi la coder avec un élément XML qui décrit sa structure mathématique sous la forme

```
<apply>
  <power/>
  <apply>
    <plus/>
    <ci>a</ci>
    <ci>b</ci>
  </apply>
  <cn>2</cn>
</apply>
```

qui décrit exactement l'arbre syntaxique de l'expression mathématique.

La plupart des éléments MathML (rappelons qu'un élément XML est la partie du fichier encadrée par une balise ouvrante et un balise fermante, et comprenant ces balises) comme `msup` s'attendent à ne contenir que d'autres éléments XML. Par contraste, les éléments de noms `mi` ou `mo` s'attendent à ne contenir que des caractères et des symboles. Les symboles mathématiques sont codés sous formes d'entités XML : ils sont précédés du symbole `&` et terminés par un poit virgule. C'est ainsi que l'analogue du symbole TeX `\alpha` sera en MathML l'entité `α`, l'analogue du symbole TeX `\cup` sera en MathML l'entité `∪`.

A partir de là, tout mathématicien ayant une pratique suffisante de TeX comprendra facilement les exemples suivants :

$$(1 \quad 2) \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = {}^{t}(2 \quad 1)$$

```
<reln> <eq/>
<apply><times/>
<vector> <cn>1 </cn> <cn>2 </cn>
</vector>
<matrix>
<matrixrow> <cn>0 </cn> <cn>1 </cn> </matrixrow>
<matrixrow> <cn>1 </cn> <cn>0 </cn> </matrixrow>
</matrix>
</apply>
<apply> <transpose/>
<vector> <cn>2 </cn> <cn>1 </cn>
</vector>
</apply>
</reln>
```

ou encore

$$\sum_{n=0}^{+\infty} x^n$$

```
<apply><sum/>
<bvar> <ci>n</ci> </bvar>
<lowlimit> <cn>0</cn> </lowlimit>
<uplimit> <ci>&infty;</ci> </uplimit>
<apply><power/>
<ci>x</ci>
<ci>n</ci>
</apply>
</apply>
```

Il n'échappera pas au lecteur que MathML n'est en aucune façon destiné à se substituer à TEX dans un proche avenir : aucune personne normalement constituée ne peut écrire du code MathML à la volée comme on écrit un article en LATEX. Il s'agit bien ici d'un format de fichier mathématique standardisé et non d'un traitement de textes mathématiques. Une bonne solution pour diffuser des mathématiques sur le Web (à l'exception bien entendu du PDF qui remplit un toute autre office et qui est destiné uniquement à être lu) est d'utiliser un logiciel comme TEX, Mathematica ou Maple pour écrire le texte, et ensuite de sauvegarder ce texte au format MathML, en espérant que tous les navigateurs dans un proche avenir seront capables d'afficher du MathML. Remarquons à ce propos que des fichiers XSL sont disponibles pour transformer un document MathML en un document SVG, ce qui peut être intéressant dans la mesure où il existe(ra) plus de navigateurs capables d'afficher du SVG que du MathML (chacun sait que beaucoup de personnes préfèrent les images aux formules mathématiques).

5 Références bibliographiques

5.1 Ressources XML

- "XML, Java and the Future of the Web," Jon Bosak. L'article de démarrage, au moins du point de vue du programmeur Java. Jon Bosak est considéré comme étant le père de XML :

  http ://metalab.unc.edu/pub/sun-info/standards/xml/why/4myths.htm
- Cours de XML : http ://www.w3schools.com/xml/
- Cours sur les DTD : http ://www.w3schools.com/dtd/
- "Media-Independent Publishing : Four Myths about XML" Jon Bosak :

  http ://metalab.unc.edu/pub/sun-info/standards/xml/why/4myths.htm
- Le site XML-SGML de Robin Cover permet de se procurer beaucoup de ressources XML :

  http ://www.oasis-open.org/cover/
- Le site XML de la W3C : http ://www.w3.org/XML/

- OASIS, le site Web de Organization for the Advancement of Structured Information Standards :
  `http ://www.oasis-open.org`
- XML.com est un site de ressources XML très important : `http ://www.xml.com`
- Le site XML d'IBM : `http ://www.software.ibm.com/xml/index.html`

5.2 XML et Java

- "XML and Java : The Perfect Pair" par Ken Sall (Internet.com, November 1998) :
  `http ://wdvl.com/Authoring/Languages/XML/Java/index.html`
- Le site de Sun sur XML : `http ://java.sun.com/xml/downloads/jaxp.html`

5.3 Tutoriels et apprentissage

- Cours de XML : `http ://www.w3schools.com/xsl/`
- Le Microsoft's Site Builder Network avec une série d'article intitulés "Extreme XML", très orienté Windows
  `http ://www.microsoft.com/sitebuilder/magazine/xml.asp`
- Webmonkey a une bonne série d'articles d'introduction à XML :
  `http ://www.hotwired.com/webmonkey/xml/ ?tw=xml`
- "What the ?xml !" par L.C. Rees : `http ://www.geocities.com/SiliconValley/Peaks/5957/wxml.html`
- "The XML Revolution" par Dan Connolly : `http ://helix.nature.com/webmatters/xml.html`

5.4 Cascading Style Sheets

- la page CSS du W3C : `http ://www.w3.org/Style/CSS/`
- Cours de CSS : `http ://www.w3schools.com/css/`
- "Cascading Style Sheets Designing for the Web" par Hakom Wium Lie and Bert Bos (Addison-Wesley, 1997) exemples de chapitres tirés de leur ouvrage : `http ://www.awl.com/cseng/titles/0-201-41998-X/liebos/`

5.5 Extensible Style Language (XSL)

- Le processeur XSLT xalan à télécharger : `http ://xml.apache.org/xalan-j/index.html`
- Les spécifications de XSLT, XPath et XSL peuvent être trouvées à : `http ://www.w3c.org/Style/XSL/`
- Cours de XSL : `http ://www.w3schools.com/xsl/`
- Apprendre à utiliser les processeurs XSLT en utilisant l'API JAXP : `http ://java.sun.com/xml/jaxp/index.html`
- La référence XSLT : `http ://www.mulberrytech.com/quickref/index.html`
- XSLT vu par JavaWorld : "XML Document Processing in Java Using XPath and XSLT," André Tost (September 2000) : `http ://www.javaworld.com/javaworld/jw-09-2000/jw-0908-xpath.html`
- Index d'articles supplémentaires sur Java et XML :
  `http ://www.javaworld.com/channel_content/jw-xml-index.shtml`
- "The Extensible Style Language : Styling XML Documents" (WebTechniques Magazine) tutoriel XSL et exemples : `http ://www.webtechniques.com/features/1999/01/walsh/walsh.shtml`
- Le site Microsoft pour XML et XSL : `http ://www.microsoft.com/xml`
- Un processeur XSL par James Clark à télécharger : `http ://www.jclark.com/xml/xt.html`

5.6 Simple API for XML (SAX)

- Tout sur SAX : `http ://www.megginson.com/SAX/index.html`

5.7 Document Object Model (DOM)

- La page d'information du W3C sur le Document Object Model : `http ://www.w3c.org/DOM/`
- Les recommandations DOM du W3C : `http ://www.w3.org/TR/REC-DOM-Level-1/`
- Un tutoriel DOM par William Robert Stanek tiré de PC Magazine Online in "Object-Based Web Design." : `http ://www8.zdnet.com/pcmag/pctech/content/17/13/tf1713.001.html`

5.8 Scalable Vector Graphics

- Spécification de SVG : `http ://www.w3.org/TR/SVG/`

5.9 MathML

- Spécification de MathML : `http ://www.w3.org/TR/REC-MathML`
- Cours simplifié de MathML : `http :www.webeq.com/mathml`