

XSLT ET XPATH

AU SERVICE DE VOS DONNÉES XML

Guy de Pourtalès, Architecte EPFL, Chef de projet Serpentine, depourtales@vtxnet.ch

INTRODUCTION

XML, XSLT, XPath, XQuery, SVG... bien des noms qui monopolisent les conversations de tout informaticien un peu branché. Mais, au fond, à quoi tout cela peut bien servir ?

Cet article n'a aucunement l'ambition de faire de vous des maîtres du «X+quelque-chose». La seule prétention qu'il a, peut-être, est de montrer comment l'intégration de plusieurs de ces techniques peut vous aider à manipuler des données et à les représenter selon les besoins, ceci de manière simple et avec des outils aussi standards que Internet Explorer.

Malgré cela, vous aurez tout de même droit à un bref historique du XML, ainsi que les bases de sa définition. Après cette introduction et la présentation des outils utilisés pour les explications, un exemple pratique (et très basique) de transformation d'un XML (*eXtensible Markup Language*) en un SVG (*Scalable Vector Graphics*) servira de guide pour la présentation de fonctions XSL (*eXtensible Stylesheet Language*).

BREF HISTORIQUE

En 1986 le premier pas vers une description généralisée de la structure des données apparaît sous la forme du SGML (*Standard Generalized Markup Language*), norme ISO 8879:1986. Cette norme définit en effet un langage qui permet de définir de manière flexible et cohérente l'organisation et la structuration de données textuelles et binaires: la DTD (*Document Type Definition*). Celle-ci apporte un concept essentiel au sujet qui nous préoccupe: la structuration des données par balises. Les premiers utilisateurs ont été en particulier les bibliothèques qui ont pu constituer et échanger ainsi des catalogues d'ouvrages importants. Mais l'apparente complexité de son utilisation et de sa mise en place n'a pas trouvé un large écho.

Neuf ans après, soit en 1995, le mythique HTML (*Hypertext Markup Language*) voit le jour. A l'aide d'une DTD SGML, le HTML définit un langage accessible permettant la représentation ordonnée de données. Le HTML ajoute également un concept clé: les liens hypertextes, qui font le bonheur de tous les internautes frénétiques. Dans tout traitement de texte, le concept de modèle apporte un gain de temps pour la mise en page, mais surtout assure une cohérence graphique entre les documents produits. Les sites créés en HTML devenant de plus en plus importants et monopolisant de plus en plus de personnes pour leur gestion, il fallait un équivalent au HTML. C'est pourquoi, en 1996, les CSS (*Cascading Stylesheets*) vont venir compléter les outils à disposition.

Le HTML est un langage qui permet de structurer les données, mais de manière essentiellement graphique. La structure des données devient en effet évidente une fois affichée correctement dans le visualiseur adéquat. Pour répondre à ce manque (qui n'en était pas réellement un, au vu de l'existence du SGML!), le W3C (*World Wide Web Consortium*) a édité la recommandation XML en 1998. Dans sa première version, le XML utilise en effet la DTD pour définir la structure obligatoire d'un document. Toutefois, afin que cette structure puisse être définie en XML, le W3C a développé le XMLSchema appelé à remplacer la DTD.

Comme le HTML bénéficie des CSS pour la création de modèles cohérents et réutilisables, le W3C a introduit le XSLT (*eXtensible Stylesheet Language for Transformations*) qui permet de transformer les données d'une structure XML à une autre. XSLT est indissociable de XPath qui définit les règles d'accès et de navigation dans une structure XML. Comme nous le verrons plus loin, XPath s'intègre directement dans XSLT.

Finalement, en 2001 apparaît la spécification XSL (*eXtensible Stylesheet Language*) qui réunit en son sein XSLT, XPath et XSL:FO (*eXtensible Stylesheet Language: Formatting Objects*). Ce dernier langage répond à la nécessité de compléter la chaîne de processus de sélection, transformation et présentation de données structurées. XSLT et XPath, conjointement, permettent de sélectionner et modéliser des données alors que XSL:FO permet de représenter les résultats dans une mise en page prédéfinie. Actuellement, le format de sortie privilégié de XSL:FO est le PDF (*Portable Document Format*).

Il faut noter qu'à l'heure actuelle aucun moteur de traitement ne supporte entièrement et strictement la recommandation XSL. Toutefois, certains moteurs ajoutent des fonctionnalités supplémentaires de leur cru.

En fin d'article, vous trouverez les liens vers les spécifications valables à l'heure de la rédaction de cet article.

COMMENT FONCTIONNE XSL

A partir de maintenant, nous allons nous concentrer sur XSLT et XPath et faire l'impasse, à dessein, sur XSL:FO, qui mérite un article complet à lui seul. Ceci étant dit, le fonctionnement d'une transformation ressemble à n'importe quel processus de traitement de données. Un moteur XSL reçoit un fichier XML en entrée, un fichier XSLT en paramètre et produit un ou plusieurs fichiers XML, chaque fichier XSLT peut faire appel à d'autres fichiers XSLT ou inclure des fichiers XML supplémentaires, comme le montre le schéma 1.

Les fonctions XSLT vous seront présentées au fur et à mesure de notre démonstration. Mais avant de se lancer dans

notre première transformation, il faut passer par une brève présentation de ce qu'est XPath (et accessoirement, à quoi il sert!).

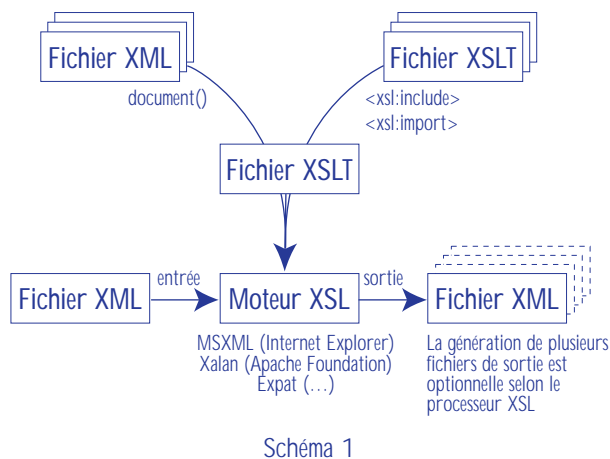


Schéma 1

Pour se repérer dans une structure XML depuis n'importe quel emplacement dans l'arbre (contexte), XPath définit six axes d'accès comme le montre le schéma ci-après. À l'aide des références à ces axes, vous pouvez accéder à n'importe quelle partie de votre structure XML.

namespace
(espace de nommage)

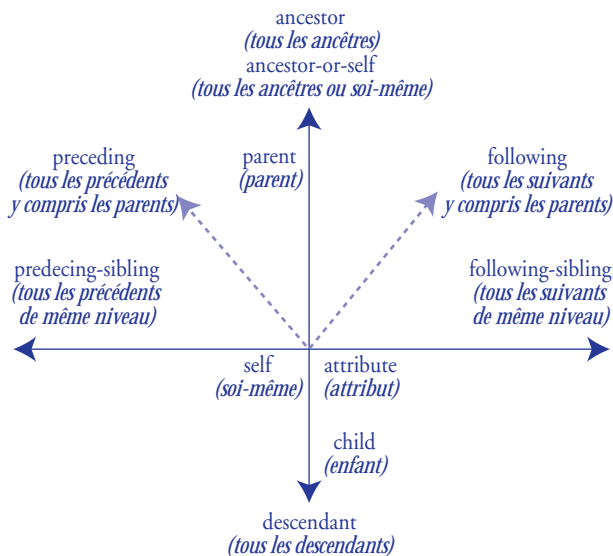


Schéma 2

En outre, XPath offre une série de fonctions de calcul, de traitement et de sélection des nœuds fort utiles dont nous utiliserons une partie par la suite.

Avant de procéder à une première réelle transformation, nous allons étudier une première transformation XSLT qui produit invariablement le même résultat, quel que soit le fichier XML traité:

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

  <xsl:template match="/">
    <html><body>
      Une transformation inutile
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

Le résultat sera toujours le document XHTML (pas très passionnant) suivant:

```
<html><body>
Une transformation inutile
</body></html>
```

Malgré son apparente inutilité, la définition de la XSLT permet de poser quelques règles de base:

ligne 1: un document XSLT est un document XML et doit comporter cet en-tête

ligne 2: un document XSLT définit toujours une balise racine `<xsl:stylesheet>` dont la référence de l'espace de nommage (`xmlns:xsl`) est située sur le site du Consortium W3.

ligne 3: un document XSLT peut définir le type de document qu'il produit. Ceci est important dans le cas de traitements nécessitant la définition du type MIME du document résultant.

ligne 5 à 9: un document XSLT définit des méthodes (`<xsl:template>`) appliquées à des nœuds. Dans ce cas, la méthode est appliquée au nœud racine («/»).

ligne 6 à 8: toutes les balises et les textes situés hors des balises de l'espace de nommage «xsl:» sont dirigés vers le document de sortie.

En résumé, notre XSLT a le comportement suivant: pour le nœud racine rencontré, écrire les balises HTML et le texte.

DANS LE VIF DU SUJET

Le système Serpentine est un transport public entièrement automatisé constitué de trois composants principaux: les capsules (véhicules), la piste en réseau (transmission d'énergie, guidage) et le système de gestion (*Traffic Manager HB®*). Ce dernier assure l'automatisation des déplacements individuels d'un ensemble de véhicules selon la demande des clients et la répartition de la charge sur le réseau. Le système de gestion s'appuie sur un grand nombre de données au format XML qui nourrissent les algorithmes sous-jacents. Dans le cas de simulations pour le dimensionnement des réseaux ou dans celui de la surveillance des mouvements des capsules, il est indispensable de représenter ces données et les résultats des calculs sous forme graphique. Pour ce faire, nous utilisons une XSLT qui produit un fichier SVG. Nous utilisons Xalan de la Fondation Apache comme moteur XSL, en particulier dans le cadre d'un serveur Jakarta Tomcat avec Cocoon comme gestionnaire de transformation XSL. Pour visualiser les SVG, nous utilisons soit le plug-in SVGViewer d'Adobe, soit le paquet Batik de la Fondation Apache.

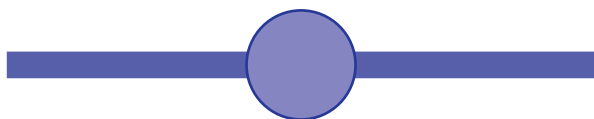
Jetons tout d'abord un coup d'œil sur un extrait des données XML que nous aurons à traiter:

```
<?xml version="1.0"?>
<MELFIN>
  <ELFIN ID_G="200209292309436"
  ID="200209301053265">
    <FORME>
      <GEOGRAPHIE>
        <POINT>0.0,0.0</POINT>
        <POLYLIGNE>
          1,-10.0,0.0
          2,10.0,0.0
        </POLYLIGNE>
      </GEOGRAPHIE>
    </FORME>
```

```
</ELFIN>
</MELFIN>
```

L'objectif de cet article est de montrer comment récupérer la coordonnée X du point pour la placer comme centre du cercle représentant le point.

Venons en au du SVG modèle que nous devons obtenir:



```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102/
/EN" "http://www.w3.org/TR/CR-SVG-20001102/DTD/
svg-20001102.dtd">
<svg>
  <g transform="translate(0,0)" style="fill:red;
stroke:blue; stroke-width:01;">
    <circle cx="0.0" cy="0.0"
           r="2" id="200209301053265"/>
    <polyline style="stroke-width:1;"
             points="-10.0,0.0 10.0,0.0"/>
  </g>
</svg>
```

La coordonnée X récupérée doit se placer comme valeur de l'attribut cx de la balise <circle>.

Maintenant que la source et la cible sont identifiées, nous pouvons commencer à construire notre XSLT:

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

  <xsl:output="xml"
doctype-public=" "-//W3C//DTD SVG 20001102//EN"
doctype-system=" http://www.w3.org/TR/CR-SVG-
20001102/DTD/svg-20001102.dtd"/>

  <xsl:template match="/"/>
</xsl:stylesheet>
```

Dans cette première étape, nous définissons le type de document généré, soit un document XML répondant à la DTD SVG référencée sur le site du W3C. Toutefois, notre XSLT ne traite pour l'instant aucune donnée.

Afin de définir les balises principales du SVG, nous faisons simplement correspondre les balises principales du SVG (soit <svg> et <g>) à la balise racine de notre XML:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
  <xsl:output="xml"
doctype-public=" "-//W3C//DTD SVG 20001102//EN"
doctype-system=" http://www.w3.org/TR/CR-SVG-
20001102/DTD/svg-20001102.dtd"/>
  <xsl:template match="/">
    <svg>
      <g transform="translate(0,0)" style=
"fill:red;stroke:blue;stroke-width:01;">
        </g>
      </svg>
    </xsl:template>
  </xsl:stylesheet>
```

Le résultat comme attendu définira les balises SVG telles quelles dans le document de sortie:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102/
/EN" "http://www.w3.org/TR/CR-SVG-20001102/DTD/
svg-20001102.dtd">
```

```
<svg>
  <g transform="translate(0,0)" style=
"fill:red; stroke:blue; stroke-width:01;">
  </g>
</svg>
```

Comme l'élément (la coordonnée X) qui nous intéresse se trouve dans une balise <POINT>, nous allons définir une méthode s'appliquant spécifiquement à cet élément et y faire appel depuis le traitement du noeud racine:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
  <xsl:output="xml" doctype-public=" "-//W3C//DTD
SVG 20001102//EN"
doctype-system=" http://www.w3.org/TR/
CR-SVG-20001102/DTD/svg-20001102.dtd"/>
  <xsl:template match="/">
    <svg>
      <g transform="translate(0,0)"
style="fill:red; stroke:blue;
stroke-width:01;">
        <xsl:apply-templates select="MELFIN/
ELFIN/FORME/GEOGRAPHIE/POINT">
          </g>
        </svg>
      </xsl:template>

      <xsl:template match="MELFIN/ELFIN/FORME/
GEOGRAPHIE/POINT">
        <circle cx="2" cy="2" r="2"/>
      </xsl:template>
    </xsl:stylesheet>
```

Une méthode peut s'appliquer spécifiquement à un noeud de l'arbre XML. Son accès s'effectue toujours relativement au contexte dans lequel se trouve le processeur. Dans notre cas, comme nous voulons appeler la méthode depuis le contexte racine du traitement, nous devons spécifier le chemin complet jusqu'à la balise qui nous concerne. Ceci est une première application de XPath. Notez que, volontairement, les valeurs cx et cy sont fausses.

L'appel à une méthode s'effectue à l'aide de l'ordre <xsl:apply-templates select="chemin/du/noeud/relatif/au/contexte">. Si le noeud <POINT> n'existait pas, l'appel ne s'effectuerait simplement pas.

Voici donc le résultat:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102/
/EN" "http://www.w3.org/TR/CR-SVG-20001102/DTD/
svg-20001102.dtd">
<svg>
  <g transform="translate(0,0)" style="fill:red;
stroke:blue; stroke-width:01;">
    <circle cx="2" cy="2" r="2"/>
  </g>
</svg>
```

Dans le cas qui nous préoccupe, nous voulons modifier la valeur de l'attribut cx de la balise <circle>, c'est pourquoi nous devons sortir l'attribut de la balise afin de pouvoir lui assigner une valeur:

```
<xsl:template match="MELFIN/ELFIN/FORME/
GEOGRAPHIE/POINT">
  <circle cy="2" r="2">
    <xsl:attribute name="cx">3</xsl:attribute>
  </circle>
</xsl:template>
```

La modification dans le document est invisible, à part le changement de la valeur de `cx` de 2 à 3, preuve que l'appel à `<xsl:attribute>` a fonctionné.

Vous vous souvenez que les coordonnées `x` et `y` sont séparées par une virgule dans la définition du point de notre document source. Afin d'extraire la coordonnées `x`, nous allons devoir créer une méthode **nommée** dont la fonction sera d'extraire tous les caractères jusqu'au premier séparateur passé en paramètre. Elle recevra également en paramètre la valeur de la balise `<POINT>` soit le vecteur «`x,y`».

Ajoutons à notre XSLT la méthode suivante:

```
<xsl:template name="getPointX">
  <xsl:param name="coordinates"/>
  <xsl:param name="separator"/>
  <xsl:value-of select="substring-
    before($coordinates,$separator)"/>
</xsl:template>
```

Les paramètres des méthodes sont définis au début de la méthode par `<xsl:param>`. Pour faire référence à un paramètre ou une variable dans une fonction XSLT ou XPath, nous utilisons la convention `$nom_de_la_variable`.

L'ordre `<xsl:value-of>` a pour fonction d'écrire en sortie la valeur choisie par l'attribut `select`. Si, par exemple, nous avons écrit `<xsl:value-of select="MELFIN/ELFIN/FORME/GEOGRAPHIE/POINT">`, nous aurions obtenu à la sortie «0.0,0.0». Afin d'extraire le texte avant un séparateur, nous faisons appel à une fonction XPath `substring-before(texte,texte)`.

Afin d'être une dernière fois didactique, nous allons définir une variable dans notre méthode de traitement de la balise `<POINT>` qui prendra la valeur résultante de l'appel à la méthode `getPointX`:

```
<xsl:template match="MELFIN/ELFIN/FORME/
  GEOGRAPHIE/POINT">
  <xsl:variable name="x">
    <xsl:call-template name="getPointX">
      <xsl:with-param name="coordinates">
        <xsl:value-of select="."/>
      </xsl:with-param>
      <xsl:with-param name="separator">
        </xsl:with-param>
    </xsl:call-template>
  </xsl:variable>

  <circle cy="2" r="2">
    <xsl:attribute name="cx">
      <xsl:value-of select="$x"/></xsl:attribute>
    </circle>
  </xsl:template>
```

La définition d'une variable s'effectue à l'aide de l'ordre `<xsl:variable>`, sa valeur étant définie entre la balise d'entrée et de fin. Afin de récupérer la valeur `x` que nous cherchons, nous appelons la méthode `getPointX` à l'aide `<xsl:call-template>` à laquelle nous passons la valeur du contexte actuel (rappelez-vous: `<POINT>`). XPath définit le contexte actuel par «.» tout comme sur un système de fichier UNIX. En plus, nous passons une virgule comme paramètre de séparation. Remarquez que les paramètres sont nommés lors de l'appel à la méthode, l'ordre de passage n'important pas.

Pour finir, nous passons à la valeur de l'attribut `cx` la valeur de la variable `x`, pour obtenir le document suivant:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102/
/EN" "http://www.w3.org/TR/CR-SVG-20001102/DTD/
svg-20001102.dtd">

<svg>
  <g transform="translate(0,0)" style="fill:red;
    stroke:blue; stroke-width:01;">
    <circle cx="0.0" cy="2" r="2"/>
  </g>
</svg>
```

Conclusion

Le petit exemple qui a été développé ci-dessus est très loin d'être optimal. Pour être franc, il pourrait être beaucoup plus compact, mais il a été écrit avec en arrière pensée la volonté de montrer quelques possibilités de XSLT et de XPath.

XSLT et XPath sont des langages très riches et les possibilités de traitement de structures XML étendues. Les notions de tri, de copie, de prédication, de recherche, de boucle, de traitement conditionnel, etc. n'ont pas été évoquées car beaucoup trop nombreuses pour entrer dans le cadre de cet article. Le meilleur conseil est d'entrer dans une librairie ou une bibliothèque et de prendre les quelques livres consacrés au sujet. En particulier, les éditions O'Reilly proposent des ouvrages très clairs et particulièrement bien structurés.

Voici encore quelques sites qui pourront vous être utiles lors de votre appréhension de ce nouveau type de traitement de données.

LES RECOMMANDATIONS DU W3C

SGML: <http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387>
 HTML: <http://www.w3.org/TR/html4/>
 XHTML: <http://www.w3.org/TR/xhtml11/>
 CSS: <http://www.w3.org/TR/REC-CSS2/>
 XML: <http://www.w3.org/TR/REC-xml>
 XSLT: <http://www.w3.org/TR/xslt20/>
 XPath: <http://www.w3.org/TR/xpath20/>
 XSL (avec XSL:FO): <http://www.w3.org/TR/xsl/>

LES SITES

- Le World Wide Web Consortium: <http://www.w3.org/>
- La Fondation Apache et leurs outils Open Source dédiés à XML: <http://xml.apache.org>
- Des didacticiels en nombre: <http://www.w3schools.com>
- Un portail d'information riche: <http://www.xml.com>
- Une collection de lien vers des outils XML gratuits: <http://www.garshol.priv.no/download/xmltools>
- Le système Serpentine <http://www.serpentine.ch> ■