

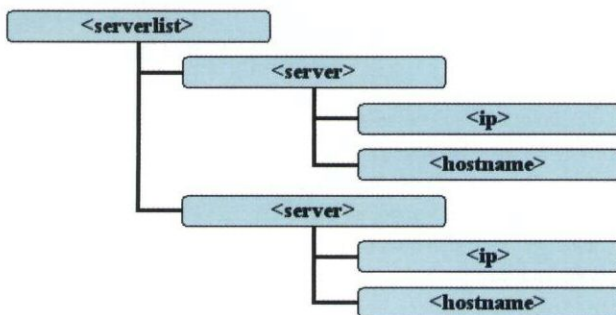
XML : les fondamentaux !

Aujourd'hui il est difficile de parler de programmation sans parler de XML. Que ce soit pour le stockage ou l'échange de données, pour la modélisation ou mille autres domaines, XML est omniprésent. Les développeurs en usent et en abusent autant pour sa simplicité d'utilisation, son format – relativement – lisible par un humain équipé d'un lecteur de texte, que pour sa pérennité. Nous allons donc détailler les causes d'un tel succès ainsi que les pièges dans lesquels ne pas tomber lorsque la question du XML se pose.

Le XML est un langage de balise, c'est-à-dire que, tout comme son ancêtre SGML, il s'agit d'un fichier texte contenant un marquage spécial (la " balise "). Ces balises délimitent des zones de données hiérarchisées dans le document. Par exemple, la représentation d'un parc de serveurs peut être faite de la façon (simpliste) suivante :

```
<serverlist>
  <server>
    <ip>192.168.0.1</ip>
    <hostname>firewall</hostname>
  </server>
  <server>
    <ip>192.168.0.2</ip>
    <hostname>web_server</hostname>
  </server>
</serverlist>
```

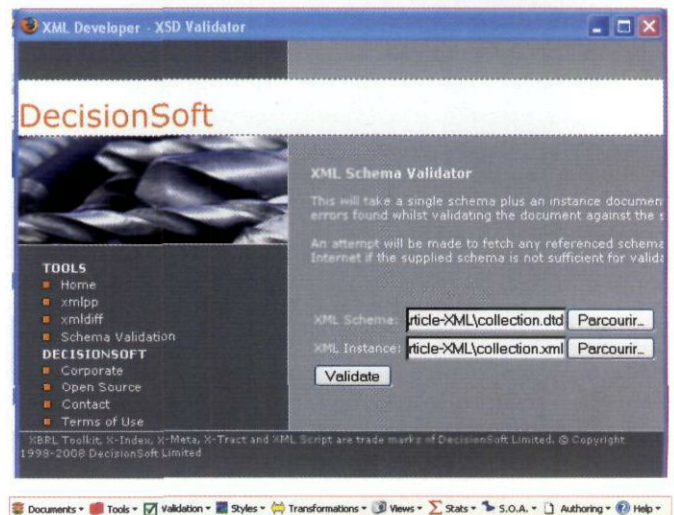
Chaque serveur est représenté dans la liste par une entité <server> qui contient elle-même une adresse IP ainsi qu'un nom d'hôte. Comme on le constate, aussi bien grâce à l'indentation qu'à la simple lecture du " code ", l'information est stockée sous une forme arborescente pouvant être représentée graphiquement de la façon suivante :



La syntaxe lisible et facilement compréhensible à la fois par un humain ou par un programme est simultanément une force et une faiblesse du XML. C'est d'ailleurs une des raisons qui a participé à son adoption massive dans le développement et particulièrement dans les applications ayant recours à des données complexes.

XML pour la gestion de données

Dans de très nombreux cas, le format XML est utilisé pour stocker des données pseudo-structurées dans un contexte potentiellement privé de base de données. Il est fréquent de rencontrer des applications proposant plusieurs méthodes de sauvegarde de données (souvent exten-



sibles par plug-in), et parmi celles-ci, il est commun de trouver un format basé sur du XML en complément d'un autre s'appuyant, par exemple, sur du SQL. On peut citer Amarak et son moteur de sauvegarde de la collection de musique.

Il est vrai que XML ne manque pas d'atouts à faire valoir en ce qui concerne ce type d'utilisation. En effet, il est une notion qui est vitale en matière de stockage de données : la pérennité des données, (les violents débats autour de la normalisation du format OOXML en témoignent...) Et, de fait, XML est un format extrêmement adapté à ce genre d'utilisation :

- un format texte ;
- peut représenter la majorité des structures de données informatiques (listes, arbres, etc.) ;
- syntaxe stricte normalisée par un comité international ;
- une structure hiérarchique adaptée à une grande partie des données à stocker ;
- indépendant de la plate-forme ;
- possible validation via un schéma ;

Parmi les exemples " célèbre " de formats de fichiers basés sur du XML, on peut citer le sulfureux OOXML, l'UML, le MathML, les formats compressés de la suite Open Office (.odt en tête) ainsi qu'une pléthore de formats ouverts. Une des conséquences induites par les avantages cités est que les documents XML sont facilement maintenables. Ainsi la compatibilité avec les versions précédentes est relativement simple à mettre en place. Prenons l'exemple, original, d'une application multimédia gérant une collection de fichiers (musique, vidéo, images). Dans un premier temps cette application ne gère qu'un fichier d'index qui ne contient rien d'autre que le nom du fichier et son répertoire. Exemple :

Programmez!

```
<?xml version="1.0" encoding="UTF-8"?>
<collection>
  <item>
    <filename>zz_top-La_Grange.mp3</filename>
    <directory>/home/arnaud/Musique</directory>
  </item>
  <item>
    <filename>Eric_Johnson-Cliffs_of_Dover.mp3</filename>
    <directory>/home/arnaud/Musique</directory>
  </item>
  <item>
    <filename>slack-get_logo.png</filename>
    <directory>/home/arnaud/Images</directory>
  </item>
  <item>
    <filename>Gekijouban_Tengen_Topppa_Gurren_Lagann.mp4</filename>
    <directory>/home/arnaud/Video</directory>
  </item>
</collection>
```

Imaginons maintenant une autre application, qui n'a pas généré le fichier de collection, mais qui l'utilise. Prenons un script automatisé qui vérifie de façon journalière la date de dernier accès de chacun des fichiers de la collection, dans le but de déplacer ceux qui n'ont pas été utilisés depuis au moins un mois dans un répertoire spécial. En Perl, ce script serait le suivant :

```
#!/usr/bin/perl
use strict;
use warnings;
use XML::Simple;
use File::Copy;
my $unused= '/home/arnaud/UnusedMedias';
die "Fichier de collection illisible ou non spécifié.\n" unless( -e $ARGV[0]
&& -r $ARGV[0] );
my $collection = XML::Simple::XMLin( $ARGV[0] , ForceArray => ['item']
, ForceContent => 1 );
foreach my $item { @{$collection->{'item'}} }{
  my $file = $item->{'directory'}->{'content'}.'/'.$item->{'filename'}->
{'content'};
  if( -e $file){
    my @stats = stat($file);
    if( (time()-$stats[8]) >= (86400*30) ){
      move($file,$unused);
    }
  }
}
```

Certains passages de ce script méritent une petite explication. Tout d'abord, le fichier de collection est passé en argument du script, puis accédé via le tableau des options @ARGV. Nous avons ensuite recours au module XML::Simple (disponible sur le CPAN) pour analyser le fichier. Ce module retourne une structure de données de type "hashref" représentant le contenu du fichier de collection. Cependant, le comportement de ce module est parfois inadapté au traitement que nous désirons mettre en place, à savoir une boucle itérative. Nous forçons

donc XML::Simple à toujours mettre les éléments <item> dans un "arrayref" (directive ForceArray => ['item']). Sans cela, s'il n'y a qu'un élément <item> dans le document, XML::Simple le dispose directement dans le hashref, ce qui ne manquera pas de provoquer un comportement désagréable lors du foreach. Cette boucle contient d'ailleurs une partie un peu ésotérique au premier abord : @{\$collection->{'item'}}. Rappelons, pour ceux qui ne sont pas coutumiers du langage Perl, que c'est une des façons possibles de déréférencer une référence sur une liste (arrayref). La fonction XMLin() se voit aussi attribué le paramètre ForceContent => 1, qui permet l'ajout d'une balise virtuelle "content" désignant le contenu d'une balise (à différencier de ses attributs). Pour finir nous testons, pour chaque fichier existant, que l'écart de temps (en secondes depuis l'epoch) entre maintenant et le dernier accès est inférieur au nombre de secondes dans un mois.

Que ce passe-t-il maintenant lors de l'évolution de l'application multimédia, particulièrement si le fichier de collection se voit enrichi de quelques balises ou attributs ? La réponse est simple : rien. Le même script exécute exactement la même besogne de la même façon et même si, par exemple, la nouvelle application génère le XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<collection>
  <item type="music">
    <filename>zz_top-La_Grange.mp3</filename>
    <directory>/home/arnaud/Musique</directory>
    <artist>ZZ Top</artist>
    <title>La_Grange</title>
  </item>
  <!-- ... -->
  <item type="movie">
    <filename video-codec="xvid" audio-codec="mp3">Gekijouban_
Tengen_Topppa_Gurren_Lagann.mp4</filename>
    <directory>/home/arnaud/Video</directory>
    <style>Japan Anime</style>
  </item>
</collection>
```

Ceci dit, tordons d'emblée le cou à une idée préconçue très répandue : ceci n'est en rien magique et le format XML a autant sa part dans ce succès que le fait que le script soit bien codé ! En effet, nous avons pris deux précautions dans l'appel au parser :

1. conditionnement de la structure de données via la directive ForceArray ;
2. différenciation explicite du contenu d'une balise et de ses attributs.

Tentez l'expérience de supprimer ces directives d'analyse pour voir... Cette compatibilité implicite est très rassurante pour le développement d'architectures lourdes où les développements sont potentiellement longs et coûteux. Une certaine sérénité s'impose naturellement quant à la pérennité des données, autant grâce à cette hiérarchisation rassurante qu'à la lisibilité de ce format textuel.

Cependant, ces possibilités seraient bien vite anéanties s'il n'était pas possible de vérifier qu'un fichier se conforme bien à un standard. C'est le rôle des fichiers de description.

Description d'un fichier XML

Un fichier XML en soi est un fichier texte, aucune force suprême ou métaphysique ne le prédispose à l'ordre et à la rigueur. Ce rôle est dévo-

Programmez!

Iu aux schémas de description du document. Ceux-ci spécifient les éléments autorisés dans le document, leur organisation hiérarchique, leur type de données, etc. Il existe de nombreux " langages " de description tels que l'historique DTD, les plus récents XML Schema et Relax NG (prononcez " relaxing ") ou encore Schematron. Nous allons nous attarder sur deux de ceux-ci : DTD et Relax NG.

En premier lieu, intéressons nous à l'ancêtre DTD (pour Document Type Definition). Langage de description lui aussi dérivé du SGML, il possède l'avantage de pouvoir être embarqué en ligne dans un document XML. Il est en outre standardisé depuis la norme XML 1.0. De plus, il est très ancré par tradition dans les développements XML. Ces menus avantages sont, d'un point de vue technique, bien loin de compenser les monstrueux inconvénients qui s'y rattachent. Citons par exemple l'absence de support des namespaces, la difficulté de mise en place des choix conditionnels, ou encore, le fait que la syntaxe DTD ne soit pas XML, ce qui complique largement la lecture de tels documents.

Voyons la définition DTD de la dernière version de la collection :

```
<!ELEMENT collection (item+)>
<!ELEMENT item (filename,directory,artist?,title?,style?)>
<!ATTLIST item type (music|video|image|unknow) "unknow">
<!ELEMENT filename (#PCDATA)>
<!ATTLIST filename audio-codec CDATA #IMPLIED >
<!ATTLIST filename video-codec CDATA #IMPLIED >
<!ELEMENT directory (#PCDATA)>
<!ELEMENT artist (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT style (#PCDATA)>
```

Un élément est déclaré via <!ELEMENT> et un attribut via <!ATTLIST>. L'élément possède des enfants qui doivent apparaître dans l'ordre dans lequel ils sont déclarés (si l'ordre n'est pas important on remplacera alors les virgules par des " pipe ". Le mot clé #IMPLIED signifie l'aspect facultatif de l'élément ou attribut.

On remarque là que nous sommes assez loin de la syntaxe hiérarchique du XML. Adonnons nous maintenant au même exercice avec la syntaxe compacte de Relax NG :

```
element collection { item+ }
element item {
  attribute type {"music"|"video"|"image"|"unknow"},
  element filename {
    text,
    attribute video-codec {text},
    attribute audio-codec {text},
  },
  element directory {text},
  {
  {
    element artist {text},
    element title {text},
  }|
  element style {text},
  }
  }?
}
```

Bien que plus verbeux que son ancestral homologue, la syntaxe RNC (Relax NG Compact) est bien plus compréhensible. On peut ensuite remarquer l'ajout d'une information de choix entre les balises <artist>, <title> et <style>.

En effet, le premier couple n'est jamais présent en même temps que <style> car ils sont relatifs à des types d'items différents. Il ne reste plus qu'à transformer ce fichier RNC en fichier RNG (la syntaxe étendue en XML) avec un outil comme Trang par exemple, puis de valider le document avec un outil comme Jing (ou la libxml2 qui fait ça très bien et qui dispose de nombreux binding vers tous types de langages de scripts). Cette transformation nous amène tout naturellement vers les feuilles de style XSLT...

XSLT : la vraie force du XML

Souvent négligées, les feuilles XSL sont pourtant LA raison qui devrait influencer le choix du XML. En effet, ces feuilles de styles sont de véritables filtres à tout faire pour un document XML. Capables de transformer n'importe quel document vers n'importe quel format le plus simplement du monde. C'est ce qui fait du XML le langage d'échange de données par excellence. Prenons notre fichier de collection par exemple, pour le transformer en SQL, il nous suffit d'écrire le fichier XSL suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:for-each select="collection/item">
      INSERT INTO Collection VALUES( <xsl:value-of select="filename"/>,"
      <xsl:value-of select="directory"/>,"<xsl:value-of select="."/ @type"/>");
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Ce qui signifie que pour chaque élément item (enfant de collection) le processeur génère une ligne SQL avec les valeurs des balises filles de l'item. La sortie générée est du type :

```
INSERT INTO Collection VALUES("zz_top-La_Grange.mp3" ,"/home/
arnaud/Musique", "music ");
```

Il y a suffisamment à dire sur les XSLT pour faire un article unique, je vous invite donc à vous documenter pour en savoir plus sur ce sujet qui mérite réellement toute votre attention.

Conclusion

Le XML est un format qui a de très nombreux avantages, cependant il est crucial de ne pas l'utiliser hors de son périmètre. La première question à poser au moment du choix est : " comment peut-on faire sans XML ? " suivi de très près par " quel impact sur l'évolutivité et les performances ? ". Typiquement, il est souvent possible de se passer de XML quand les données ne sont pas hiérarchisées. Par contre, quand il s'agit de trouver une solution d'échange/transformation de données, la question ne se pose même pas : foncez !

■ **Arnaud Dupuis**

Consultant Uperto (Open Source Business Unit) - Devoteam group