Inclure des données calculées dans un document XML avec TAL

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 mai 2007

https://www.bortzmeyer.org/xml-et-donnees-calculees-avec-tal.html

---

On a souvent besoin de mettre dans un document des données qui sont calculées, par exemple les unes par rapport aux autres (« trois serveurs à 1500 € font 4500 € ») soit extraites du monde extérieur, par exemple une base de données. XML n'est pas un langage de programmation, uniquement un langage de description de données et il faut donc compter sur d'autres systèmes comme TAL.

Reprenons l'exemple de notre document décrivant les achats de serveurs informatiques. Mettons qu'il soit rédigé en Docbook. Si j'écris :

```
<para>Nous allons donc acheter 3 PC. 3 serveurs
à 1500 &#x20ac; font 4500 &#x20ac;.</para>
```

et que je modifie le prix d'un serveur parce que j'ai obtenu une réduction, le total ne va pas être recalculé automatiquement, le fichier XML étant statique. Avec OpenOffice, je suppose qu'on mettrait les chiffres dans le tableur et qu'on ferait un lien vers le tableur depuis le document texte. Mais il existe des tas de bonnes raisons d'utiliser plutôt Docbook et je n'ai donc pas cette possibilité. Une des approches serait de faire générer le fichier XML par un programme <https://www.bortzmeyer.org/creer-xml-par-programme.html>. Mais cela veut dire qu'au lieu de travailler avec un fichier XML relativement lisible par tous, on va travailler avec un source Perl ou Java, ce qui n'est certainement pas un progrès, d'autant plus que la partie calculée n'est qu'une faible portion de tout le document.

Une autre approche est d'ajouter, grâce au langage TAL quelques petits éléments calculés dans le document XML. TAL avait originellement été inventé pour Zope mais existe pour d'autres environnements. Sa grande force (par rapport à d'autres préprocesseurs) est qu'un document XML comportant du TAL reste un document XML et peut donc être manipulé par des outils XML. Mieux, TAL n'utilise que des **attributs** XML ce qui limite le risque de conflit avec, par exemple, des éditeurs XHTML qui pourraient s'étonner de rencontrer des élements XML inconnus.

Prenons donc un simple fichier Docbook avec du TAL :

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN" "http://www.oasis-open.org/docbook/xml/4.4/doc
<article xmlns:tal="http://xml.zope.org/namespaces/tal">
<title>Achats de serveurs</title>
<para>Nous allons donc acheter <phrase tal:content="num_servers"/>
PC. <phrase tal:content="num_servers"/> serveurs à <phrase
tal:content="price_server"/> &#x20ac; font <phrase
tal:content="python: num_servers*price_server"/> &#x20ac;.</para>
</article>
```

Pour transformer ce document en pur Docbook, nous allons utiliser l'implémentation SimpleTal <http://www.owlfish.com/software/simpleTAL/> en Python.

```
from simpletal import simpleTAL, simpleTALES
import sys, os, re, logging

context = simpleTALES.Context(allowPythonPath=True)
simpleTALLogger = logging.getLogger ("simpleTAL")
simpleTALESLogger = logging.getLogger ("simpleTALES")
simpleTALLogger.setLevel (logging.ERROR)
simpleTALESLogger.setLevel (logging.ERROR)

# Les variables sont mises "en dur" dans le programme. Naturellement,
# dans la réalité, elles seraient plutôt lues dans un fichier ou une
# base de données.
context.addGlobal ("num_servers", 3)
context.addGlobal ("price_server", 1500)

if len(sys.argv) <= 1:
    sys.stderr.write("Usage: %s file...\n" % sys.argv[0])
    sys.exit(1)

for filename in sys.argv[1:]:
    templateFile = open (filename, 'r')
    template = simpleTAL.compileXMLTemplate (templateFile)
    templateFile.close()
    resultfile = open (re.sub ("_tal$", "", filename), "w")
    template.expand (context, resultfile, outputEncoding="UTF-8")
```

Ce programme peut s'exécuter ainsi :

```
% python generate.py achat-servers.db_tal
```

et donne bien un fichier Docbook pur contenant les données calculées.

Si on veut valider le document Docbook (une bonne idée), le mieux (et le plus simple) est sans doute de le valider après la transformation en Docbook pur (ainsi, les effets de la transformation du TAL seront pris en compte). Toutefois, si on veut le valider avant, il faut utiliser une variante du schéma Docbook, prenant en compte les attributs TAL, par exemple, en Relax NG :

```
namespace tal = "http://xml.zope.org/namespaces/tal"

tal-define.attrib = attribute tal:define { text }?
tal-attributes.attrib = attribute tal:attributes { text }?
tal-condition.attrib = attribute tal:condition { text }?
tal-content.attrib = attribute tal:content { text }?
```

```
tal-replace.attrib = attribute tal:replace { text }?
tal-repeat.attrib = attribute tal:repeat { text }?
tal-on-error.attrib = attribute tal:on-error { text }?
tal-omit-tag.attrib = attribute tal:omit-tag { text }?
Tal.attrib =
       tal-define.attrib,
        tal-attributes.attrib,
        tal-condition.attrib,
        tal-content.attrib,
        tal-replace.attrib,
        tal-repeat.attrib,
        tal-on-error.attrib,
        tal-omit-tag.attrib

local.common.attrib &= Tal.attrib
```