

LA GÉNÉRATION DE DOCUMENTS PDF depuis un serveur applicatif



ALEXANDER.LAMB@dim.hcuqe.ch &
NICOLAS.CASSONI@dim.hcuqe.ch,
HÔPITAUX UNIVERSITAIRES DE GENÈVE



INTRODUCTION

Dans bien des applications de type Web, il est nécessaire de générer des documents au format PDF (www.adobe.com/products/acrobat/adobe.pdf.html). En effet bien que l'interface utilisateur de type HTML soit parfaite lors de sessions interactives, il est important de pouvoir fournir à l'utilisateur certaines pages dans un format imprimable.

Nous allons ici rapidement passer en revue les possibilités offertes, puis détailler une solution choisie aux Hôpitaux Universitaires de Genève à la division d'informatique médicale.

PRINCIPES ET SOLUTIONS DE GÉNÉRATION DE DOCUMENTS PDF

PRINCIPES

Il y a plusieurs manières de créer un document en PDF. En effet, un document PDF n'est en fait qu'une description lisible par un être humain d'ordres de dessins (lignes, remplissages, textes, fontes, etc.). Le format PDF est d'ailleurs inspiré du Postscript, avec la possibilité de programmation

en moins. Bien que lisible (sauf dans la version encryptée) le format ne permet pas à un être humain de le générer *à la main* comme pour un fichier HTML par exemple. En effet, non seulement le format est complexe, mais un système de dictionnaire interne au format nécessite le positionnement des instructions au byte près!

Ainsi, il est possible au travers de bibliothèques spécialisées de générer directement un document PDF avec des fonctions dans le genre de **createDocument**, **addTextWithFontAtPosition**, **drawLineFromTo**, etc. Toutefois, attaquer le problème à un si bas niveau s'avère en général fastidieux. Les problèmes de calcul de sauts de pages, de positionnement des éléments graphiques, et (non des moindres) de formatage d'un bloc de texte en plusieurs lignes avec césures de mots, restant du ressort de l'application appelant ces fonctions.

Il est également possible en utilisant divers logiciels disponibles soit commercialement soit en OpenSource, de convertir des pages du format HTML au format PDF. Toutefois, cette solution,

SUITE EN PAGE 7

SOMMAIRE FI 1/2003

- 1 La génération de documents PDF depuis un serveur applicatif
- 2 Cellule AFS epfl.ch en service public
- 2 Propriétaires de noms de domaines: ayez l'oeil sur vos contrats!
- 2 Poste pour jeune chercheur
- 3 Réseaux IP – Voix et multi-média sur IP
- 11 Un projet pionnier de Voice over IP (VoIP) au Centre Cantonal des Télécommunications (CCT)
- 13 *iGames* – Implémentation d'un service de jeu en réseau employant les agents mobiles
- 17 Programme des cours
- 21 FileMaker Pro 5.5 ou 6.0 – Sésame ouvre-toi !
- 23 SWITCH en vitesse
- 24 Calendrier

PROCHAINES PARUTIONS

	délai RÉDACTION	PARUTION
2	06.02.03	25.02.03
3	06.03.03	25.03.03
4	03.04.03	29.04.03
5	15.05.03	03.06.03
6	19.06.03	08.07.03
SP		19.08.03
7	28.08.03	16.09.03
8	02.10.03	21.10.03
9	30.10.03	18.11.03
10	27.11.03	16.12.03

SUITE DE LA PREMIÈRE PAGE

bien que simple à mettre en œuvre pose deux problèmes: tout d'abord le format HTML étant moins riche que le format PDF, la conversion ne présente souvent pas beaucoup d'intérêt, hormis peut-être la génération automatique d'en-têtes et pieds de page. Ensuite, pour que ces logiciels fonctionnent, il s'agit en général de créer un fichier (au format HTML) qui sera ensuite converti dans un autre fichier (au format PDF). Le serveur applicatif devant donc générer des fichiers avec des noms uniques là où a priori un travail en mémoire aurait suffi, doit relire ceux-ci pour les envoyer au poste client, et finalement doit gérer l'effacement de ces fichiers temporaires.

Il existe une troisième solution qui consiste à créer un fichier chablon dans un méta langage qui permettra à une librairie de génération PDF de haut niveau de prendre celui-ci en entrée, le fusionner avec les données et produire le résultat en PDF, souvent directement en mémoire prêt à être renvoyé sur le poste client.

Nous n'aborderons pas ici les solutions d'édition à partir de bases de données. En effet, il existe des outils permettant de lancer des requêtes complexes en SQL et générer le résultat, souvent en PDF. Utiliser ce genre d'outil suppose tout d'abord que toutes les informations nécessaires à générer les documents sont contenues dans une base de données mais surtout nécessite une réécriture d'une partie de la logique métier nécessaire en SQL (ou dérivé) alors que celle-ci était déjà présente dans l'application. Ceci avec la conséquence évidente d'augmenter la quantité de travail, le risque d'erreurs ou d'incohérences et l'effort de maintenance.

SOLUTIONS

Nous allons aborder ici les solutions qui tombent dans la catégorie permettant la création de fichiers chablons et la génération en mémoire des documents.

Sachant que nous utilisons le serveur applicatif WebObjects(tm) (www.apple.com/webobjects) et par conséquent le langage Java (jdk 1.3.1 ou plus récent), un rapide tour non exhaustif du marché (libre ou non) nous a permis de référencer les solutions suivantes:

- ReportMill (www.reportmill.com)
- PDFGenerator (www.cluster9.com)
- iText: (sourceforge.net/projects/itext/)
- RReport: (www.java4less.com/print_java_e.htm)
- FOP: (xml.apache.org/fop/)
- SVGObjects: (www.svgobjects.com)

ReportMill est une solution commerciale de très bonne qualité. Le produit vient avec un éditeur graphique qui permet de construire les chablons des pages et d'y connecter les sources de données (directement depuis le EOModel dans le cas d'un usage avec WebObjects(tm)). Ils appellent cela du *object reporting*. Cela signifie qu'une zone du document n'est pas forcément simplement une colonne d'une table ou un attribut d'un objet mais peut être également le résultat d'une fonction. Ainsi, la réutilisation du code de la logique métier est maximale. Récemment la société a ajouté le support pour d'autres serveurs applicatifs ainsi

que la génération du format Flash (www.macromedia.com/software/flash/) pour les applications interactives. L'utilisation du logiciel depuis WebObjects(tm) est extrêmement simple et ne requiert qu'environ 3 lignes de codes pour initialiser le document avec la source de données, déclencher la génération et retourner le résultat avec le retour de la requête utilisateur. Ce produit a toutefois un coût et ne concernera donc que les entreprises qui le récupéreront largement dans le gain en productivité.

PDFGenerator est une solution similaire que nous n'avons pas eu l'occasion de tester. C'est également une solution commerciale.

iText est une solution OpenSource. En soit il ne représente pas vraiment ce que nous recherchons puisqu'il s'agit d'une librairie d'assez bas niveau de génération de code PDF. Toutefois, le site fait mention d'Intermezzo qui permet la génération d'un document PDF à partir d'une source XML (utilisant iText), ou même de fusionner un document chablon XML avec des données d'un autre fichier pour en faire un document PDF.

RReport semble aller dans la direction voulue en offrant une encapsulation de iText. L'intérêt principal étant de cacher au développeur les finesses de calculs du positionnement. Ils ont un outil permettant de construire visuellement les chablons de documents. Financièrement, cela fonctionne sur le mode du shareware: très bon marché! Nous n'avons pas non plus testé cette solution (à tort probablement) car nous étions déjà trop avancé dans l'utilisation d'une autre technologie.

FOP (*Formatting Objects Processor*) est un projet OpenSource du groupe Apache. Il s'agit d'une solution de génération de documents (PDF ou autres) à partir d'un modèle en XSL.

SVGObjects est une librairie OpenSource utilisant FOP et le format SVG, (www.adobe.com/svg) spécialement destinée à l'usage avec WebObjects(tm).

Ces deux dernières solutions étant celles retenues dans notre groupe, nous allons y consacrer le reste de l'article.

GÉNÉRER DES DOCUMENTS PDF À L'AIDE DE FOP

Il existe 3 manières d'intégrer FOP à un serveur applicatif.

1. On peut générer un document XML contenant des données et envoyer celui-ci avec un chablon XSL vers un serveur Cocoon. Ce dernier effectue la fusion des deux documents en entrée et génère un document PDF en sortie.
2. On peut générer un document XSL au format FO contenant déjà les données et envoyer celui-ci vers un serveur Cocoon qui effectuera la génération PDF.
3. On peut utiliser une librairie locale à son application qui effectuera la conversion d'un document XSL en document PDF.

Un serveur Cocoon (xml.apache.org/cocoon/) est un logiciel s'installant avec un serveur Apache permettant de gérer la fusion (utilisant XSLT) de données XML avec un chablon XSL pour créer un document XSL au format FO. Ce dernier sera ensuite converti en PDF.

La première et la seconde solution utilisent un serveur Cocoon. La différence entre les deux est que dans la seconde, la fusion est déjà effectuée et le serveur ne s'occupe que de la génération PDF.

La troisième solution est techniquement la même que la seconde, sauf que la génération du PDF se fait directement dans le serveur applicatif à l'aide des classes FOP sans nécessiter d'aller-retour avec un serveur Cocoon, Cocoon utilisant ces mêmes classes.

Ici, nous avons successivement utilisé les 3 méthodes. La première générait des problèmes de stabilité et de temps de réponse (temps de fusion, volume du trafic). La seconde solution a été utilisée de façon transitoire en attendant de pouvoir migrer notre serveur applicatif sur une jdk récente supportant FOP. La troisième est celle que nous utilisons actuellement en production et pour laquelle nous allons détailler le fonctionnement. Nous allons également décrire les bibliothèques réutilisables que nous avons développées pour une intégration simplifiée dans nos applications WebObjects(tm).

ET SVG ALORS ?

SVG est un format défini par Adobe permettant de dessiner ou d'inclure des images dans un document. Il est probable que c'est la réponse d'Adobe au format Flash de Macromédia.

Dans le format XSL/FO, certaines instructions sont en fait du SVG. En particulier, l'inclusion d'une image dans un document se fait grâce à des balises SVG qui seront ensuite interprétées par les classes SVG correspondantes.

Finalement, pour l'intégration de cette technologie dans WebObjects(tm), nous avons utilisé le projet SVGObjects qui consiste en du code assez simple et élégant pour automatiser la génération du PDF partant du format XSL/FO. SVGObjects a été écrit par Ravi Mendis, un ancien employé d'Apple, consultant WebObjects(tm) et collègue de l'un des auteurs de ce document à l'époque où il était lui-même chez Apple. Pour plus d'informations, il est recommandé de lire *WebObjects developer's Guide* chez SAMS (Ravi Mendis).

LES BIBLIOTHÈQUES NÉCESSAIRES

Pour compiler une application utilisant XSL/FO et exécuter la génération PDF, il faut intégrer les bibliothèques java suivantes (toutes se trouvent sur le site Apache):

- Batik: interprétation du SVG
- Fop: interprétation du XSL/FO
- Logkit-1.0: nécessaire à batik et fop
- Avalon-framework: nécessaire à batik et fop

A la date de ce document, la version de FOP était 0.23. Il peut sembler à la lecture de ce nombre que nous sommes face à une bibliothèque tout juste en qualité bêta, mais pour nos besoins, aucun problème sérieux n'est venu perturber le développement. FOP étant activement développé dans le groupe Apache, on peut espérer de nouvelles versions rapidement.

EXEMPLES ET INTÉGRATION AVEC WEBOBJECTS(TM)

WebObjects(TM) (WWW.APPLE.COM/WEBOBJECTS)

WebObjects(tm) est un environnement de développement et de déploiement d'applications Web dont l'origine

remonte à février 1996, date à laquelle la société NeXT(tm) Software a présenté ce qui était probablement le premier serveur applicatif sur le marché. A l'époque, l'environnement fonctionnait avec le langage Objective-C et bénéficiait des bibliothèques et de l'expérience OpenStep(tm). Depuis, la société NeXT a été rachetée par Apple et, le marché dictant ses préférences, WebObjects(tm) (aujourd'hui en version 5.2) a été réécrit en pur Java, mais tout en gardant bien des concepts et *patterns* OpenStep(tm) qui font aujourd'hui encore l'originalité et la qualité du produit.

WebObjects(tm) permet le développement sur Windows(tm) ou MacOSX(tm). Le produit contient un environnement complet de développement et de déploiement. Etant maintenant en pur Java, il est possible de déployer les applications sur tout système supportant une jdk 1.3.1 ou mieux. Toutefois, ce sont les plates-formes MacOSX, Windows2000 et Solaris qui sont supportées d'office par le produit.

Parmi les forces du produit on peut noter EOF (*Enterprise Objects Framework*) une couche de persistance d'objets dans les bases relationnelles particulièrement efficace, la possibilité de définir des composantes HTML ou autres réutilisables sous formes d'objets, une excellente séparation de la couche métier de la couche présentation, la génération automatique de WebServices, et finalement, la possibilité de déployer des applications avec des postes client en Swing plutôt que HTML.

Même si d'un point de vue marketing, Apple n'est pas aussi présent que les autres grands noms dans le monde J2EE (WebObjects n'est pas à proprement parler un serveur J2EE, mais peut s'y intégrer), il est intéressant de noter que beaucoup d'entreprises utilisent WebObjects pour des applications Internet ou Intranet de haute disponibilité.

EXEMPLES DOCUMENTS XSL/FO

Voici quelques exemples de documents tels qu'ils doivent être générés avant leur transformation par les classes FOP en PDF. Ces exemples sont partiels et devront être adaptés avant de les essayer!

Tout d'abord la définition d'une page:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set>
  <fo:simple-page-master
    master-name="first"
    margin-top="0.5cm"
    margin-bottom="0.0cm"
    margin-left="1.5cm"
    margin-right="0.5cm"
    page-width="21cm"
    page-height="29.7cm">
    <fo:region-body
      region-name="xsl-region-body"
      margin-bottom="1.7cm"
      margin-top="1cm"
      margin-right="1cm"/>
    <fo:region-before region-name="xsl-region-before" extent="1cm" />
    <fo:region-after region-name="xsl-region-after" extent="1.7cm" />
  </fo:simple-page-master>
</fo:layout-master-set>
</fo:root>
```

Ensuite la définition d'en-têtes et de pieds de page. Commençons pas une en-tête simple:

```
<fo:static-content flow-name="xsl-region-before">
  contenu du header...
</fo:static-content>
```

Maintenant un pied de page avec calcul du numéro de la page:

```
<fo:static-content flow-name="xsl-region-after">
  <fo:table>
    <fo:table-column column-width="9cm" />
    <fo:table-column column-width="9cm" />
    <fo:table-body>
      <fo:table-row space-before.optimum="0.1cm">
        <fo:table-cell>
          <fo:block text-align="start"
                    font-size="8pt">
            Imprimé par Urlu Berlu
          </fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block text-align="end"
                    font-size="8pt">
            Page <fo:page-number/><fo:page-number-
            citation ref-id="lastBlock" />
          </fo:block>
        </fo:table-cell>
      </fo:table-row>
    </fo:table-body>
  </fo:table>
  <fo:block>
    <fo:leader leader-length="18cm"
              rule-thickness="0.1mm" color="#000000"/>
  </fo:block>
</fo:static-content>
```

Et finalement, le contenu de la page:

```
<fo:flow flow-name="xsl-region-body">
  Le contenu de la page...
  <!-- block vide avec id pour connaitre nombre
  total de page -->
  <fo:block id="lastBlock"/>
</fo:flow>
```

INTÉGRATION AVEC WebObjects(TM)

WebObjects(tm) permet la création de composantes. Ces composantes sont formées d'éléments fixes et d'éléments qui seront dynamiques, c'est à dire remplacés lors de la génération par des valeurs provenant du code Java.

Ainsi, une composante est en fait un chablon, un contrôleur écrit en Java (sous classe WOComponent de la librairie WebObjects(tm)) et un fichier définissant les liens entre les deux (tel bouton devra appeler telle fonction, telle zone devra être remplie par telle variable, etc.).

En général, dans une application WebObjects(tm) classique, les composantes sont des pages HTML ou des sous-ensemble de pages HTML (il est possible également de définir des morceaux de pages qui seront réutilisés). Toutefois, le principe du chablon et de la génération dynamique est également applicable à d'autres formats. Par exemple, nous faisons un usage important du XML et utilisons parfois ce principe pour générer des réponses comme WebServices. Ici, c'est la génération de XSL/FO qui nous intéresse!

Ainsi, notre exemple devient:

```
<WEBOBJECT NAME=hug.pdfcomponent.foundation.Layout1>
<WEBOBJECT NAME=hug.pdfcomponent.foundation.Header1></WEBOBJECT>
<WEBOBJECT NAME=hug.pdfcomponent.foundation.Footer1>
  <fo:block text-align="center" font-size="9pt">
    Hôpitaux Universitaires de Genève, Rue Micheli-du-Crest 24
  </fo:block>
  <fo:block text-align="center" font-size="9pt">Tel 022/372.33.11</fo:block>
</WEBOBJECT>
<WEBOBJECT NAME=hug.pdfcomponent.foundation.Body1>
  <!-- motif de consultation -->
  <fo:block space-before.optimum="1cm" font-size="10pt">
    <fo:block font-weight="bold">Motif de consultation:</fo:block>
    <fo:block start-indent="1cm" space-before.optimum="0.2cm">
      <WEBOBJECT NAME=PFDataDisplayComponent10></WEBOBJECT>
    </fo:block>
    <fo:block start-indent="1cm" space-before.optimum="0.2cm">
      <WEBOBJECT NAME=PFDataDisplayComponent2></WEBOBJECT>
    </fo:block>
  </fo:block>
  <!-- diagnostic retenu -->
  <fo:block space-before.optimum="0.5cm" font-size="10pt">
    <fo:block font-weight="bold">Diagnostics retenus:</fo:block>
    <fo:block start-indent="1cm" space-before.optimum="0.2cm">
      <WEBOBJECT NAME=PFDataDisplayComponent3></WEBOBJECT>
    </fo:block>
  </fo:block>
  <!-- traitements -->
  <fo:block space-before.optimum="0.5cm" font-size="10pt">
    <fo:block font-weight="bold">Traitements:</fo:block>
    <fo:block start-indent="1cm" space-before.optimum="0.2cm">
      </fo:block>
  </fo:block>
  <!-- salutations -->
  <fo:block space-before.optimum="1cm" font-size="10pt">
    Veuillez agréer, cher (chère) Collègue, nos sincères salutations.</fo:block>
  <!-- signature -->
  <fo:block space-before.optimum="2cm" font-size="10pt">
    Dr <WEBOBJECT NAME=WOSTring3></WEBOBJECT>
  </fo:block>
</WEBOBJECT>
</WEBOBJECT>
```

On voit ici un certain nombre de balises <WEBOBJECT>. Ces balises seront interprétées en temps réel et remplacées par une nouvelle chaîne de caractères. Cette dernière peut être à son tour une composante XSL/FO avec des balises WebObjects(tm). En effet, la génération est récursive.

En particulier, la balise Layout1 contient la structure de la page. Cette composante aura le même aspect que le XSL présenté plus haut pour la définition d'une page avec une balise indiquant où placer le contenu:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  ... idem que plus haut...
  <WEBOBJECT NAME=ComponentContent1></WEBOBJECT>
</fo:root>
```

Chaque balise WebObjects(tm) représente donc une sous-composante qui sera incluse à la génération. Les noms des balises correspondent aux noms des liens que l'on retrouve dans le fichier de connexions. Ce fichier à l'aspect suivant:

```
hug.pdfcomponent.foundation.Footer1: hug.pdfcomponent.foundation.Footer
{
  user = session.user;
}

hug.pdfcomponent.foundation.Layout1: hug.pdfcomponent.foundation.Layout
{

}

WOString3: WOString
{
  value = session.dataSet.lastUpdateUserInfo.fullName;
}
```

On remarque que ces connexions peuvent posséder des attributs qui seront évalués à la génération. Ces attributs sont spécifiques à chaque classe de composantes.

Des composantes plus complexes peuvent être utilisées comme par exemple des répétitions de zones (pour des listes) ou des conditionnelles (pour afficher un contenu sous certaines conditions seulement).

Dans ce cas, pour une conditionnelle, le chablon porterait du code de la forme:

```
<WEBOBJECT NAME=Conditional>le détail de l'arrêt de travail</WEBOBJECT>
```

et le fichier de connexion:

```
Conditional:WOConditional
{
  condition = composante.questionnaire.arretTravail ;
}
```

Pour une répétition d'une liste:

```
<WEBOBJECT NAME =Repetition1>le bloc à répéter</WEBOBJECT>
```

et le fichier de connexion:

```
Repetition1:WORepetition
{
  list = composante.listeTraitements ;
  item = composante.traitement ;
}
```

LA GÉNÉRATION DU PDF

Maintenant que nous avons les chablon, comment allons-nous générer le PDF ? C'est là qu'intervient le code faisant partie du projet SVGObjects. En effet, de même que dans WebObjects(tm), chaque composante hérite naturellement de la classe WOComponent, nous allons faire hériter chaque composante PDF de notre application de la classe FOComponent, elle-même héritant de la classe WOComponent.

Les composantes WebObjects(tm) ont ceci d'intéressant que lors de la génération récursive, chaque composante (page ou élément de page donc) est appelée. La fonction appelée est:

```
public void appendToResponse(WOResponse response, WOContext context)
```

La plupart du temps cette fonction n'a pas besoins d'être ré-implémentée. Cette fois pourtant, nous allons intervenir au niveau le plus haut dans la génération et juste avant de retourner la page générée (qui à ce moment n'est en fait encore que du XSL/FO) faire passer celle-ci par la génération PDF.

Le code de la fonction appendToResponse ressemble donc à ceci:

```
super.appendToResponse(response, context);
Document document = response.contentAsDOMDocument();
ByteArrayOutputStream out = new ByteArrayOutputStream();
Logger logger = Hierarchy.getDefaultHierarchy().getLoggerFor("fop");
Driver driver = new Driver();
driver.setLogger(logger);
driver.setRenderer(Driver.RENDER_PDF);
driver.setOutputStream(out);
try
{
  driver.render(document);
}
catch (Exception e)
{
  e.printStackTrace();
}
// set the PDF data
NSData dataFile = new NSData(out.toByteArray());
response.setContent(dataFile);
// set the header
response.setHeader("application/pdf", "Content-Type");
response.setHeader(String.valueOf(dataFile.length()), "Content-Length");
response.removeHeadersForKey("cache-control");
```

Il n'y aura que la composante principale (celle qui représente la page complète en PDF) qui sera une sous-classe de FOComponent. En effet, les composantes telles que Layout, Header, Footer, etc. sont directement des sous-classes de WOComponent car ce n'est pas à leur niveau que l'on désire générer le PDF.

Conclusion

Après une première période d'expérimentation, nous avons graduellement créé une librairie de composantes WebObjects(tm) XSL/FO réutilisables. En effet, devant générer beaucoup de documents qui possèdent souvent des caractéristiques très proches, il était tentant de fournir aux développeurs le moyen de rapidement les créer.

Aujourd'hui, la librairie comprend une quinzaine de composantes XSL qui, rajoutées aux composantes standards WebObjects(tm) ainsi qu'aux composantes orientées données des applications cliniques, nous accélèrent le développement, tout en permettant à des programmeurs n'ayant pas de connaissance XSL/FO de travailler efficacement.

Alexander Lamb est responsable du groupe Serveurs Applicatifs à la Division d'Informatique médicale des Hôpitaux Universitaires de Genève. Il était anciennement chez NeXT puis Apple dans la division services de ces sociétés respectives, intervenant sur des sites utilisant les technologies WebObjects(tm) et OpenStep.

Nicolas Cassoni est développeur dans ce même groupe, auteur d'une partie du code permettant la génération PDF ainsi que d'autres applications WebObjects(tm) comme la saisie de données structurée ou la gestion des droits pour les utilisateurs des applications cliniques. ■