

Extrait du Easter-eggs - Spécialiste GNU/Linux

<http://www.easter-eggs.com>

Systeme d'impression PDF « RESTful » pour le Web

- Études de cas - Développement applicatif -

Date de mise en ligne : vendredi 27 février 2009

Easter-eggs - Spécialiste GNU/Linux

Sommaire

- [Historique](#)
- [Choix techniques](#)
- [Architecture du système \(...\)](#)
- [L'interface entre le Site \(...\)](#)
- [Agrégation des données dans \(...\)](#)
- [Transformation XSLT](#)
- [Le pipeline pour les documents](#)
- [Performances](#)
- [Les petits désagréments du \(...\)](#)
- [Quelques liens utiles](#)

Le monde du Web et celui de l'impression ont décidément du mal à se rencontrer. La prise en charge de l'impression est bien souvent très problématique dans le cadre d'un projet Web.

Dans la plupart des cas, il est possible de s'en sortir en utilisant les possibilités offertes par les feuilles de style CSS, mais le rendu final du document n'est pas garanti : les différents navigateurs interprètent la feuille de style selon leur humeur et cela ne fonctionne pas dans le cas de documents complexes.

Dans ce cas, l'unique solution est de produire un document au format PDF, unique garantie pour une impression de qualité. Je vais donc décrire dans cet article la solution que nous avons mise en place dans le cadre du projet [MAVISE](#).

Historique

La base de données MAVISE fournit les données sur l'ensemble des chaînes de télévision accessibles dans l'Union européenne. Elle a été développée par Easter-eggs en collaboration avec [l'Observatoire européen de l'audiovisuel \(OEA\)](#) pour le compte de [la Direction Générale de la Communication de la Commission Européenne](#).

Dans la première phase du projet, nous avons développé un système d'impression avec des feuilles de style. Ce système ne donnait pas entièrement satisfaction au client.

En effet, les pages du projet étant très complexes, le résultat de l'impression produite par les différents navigateurs était très aléatoire (coupure en plein milieu d'un tableau, impression sur plusieurs pages et sauts de pages inexplicables).

Le résultat n'était pas professionnel et il était difficile pour le client de produire un rapport papier avec les contenus de la base. Nous avons donc proposé de mettre en place un export PDF pour l'ensemble des éléments de l'application.

Choix techniques

Le travail pour produire des documents PDF consiste dans la plupart des cas à dessiner, à l'aide de différentes API, le contenu du document (texte, tableaux, graphiques et mise en page).

Les outils utilisés habituellement sont :

- [ReportLab](#) en Python ;
- [FPDF](#) en PHP ;
- [PDF::API2](#) en Perl.

Il s'agit d'un travail long et fastidieux, qui consiste à fournir une suite d'instructions pour former le document.

Cette technique montre vite ses limites :

- les demandes de modification de mise en page du document produit sont lourdes et nécessitent beaucoup de temps ;
- le code pour produire le document est difficilement rationalisable ;
- elle ne convient pas à des documents de grande taille.

Dans le modèle [MVC](#), qu'utilisent la plupart des applications Web, nous utilisons un système de template pour l'affichage (la vue).

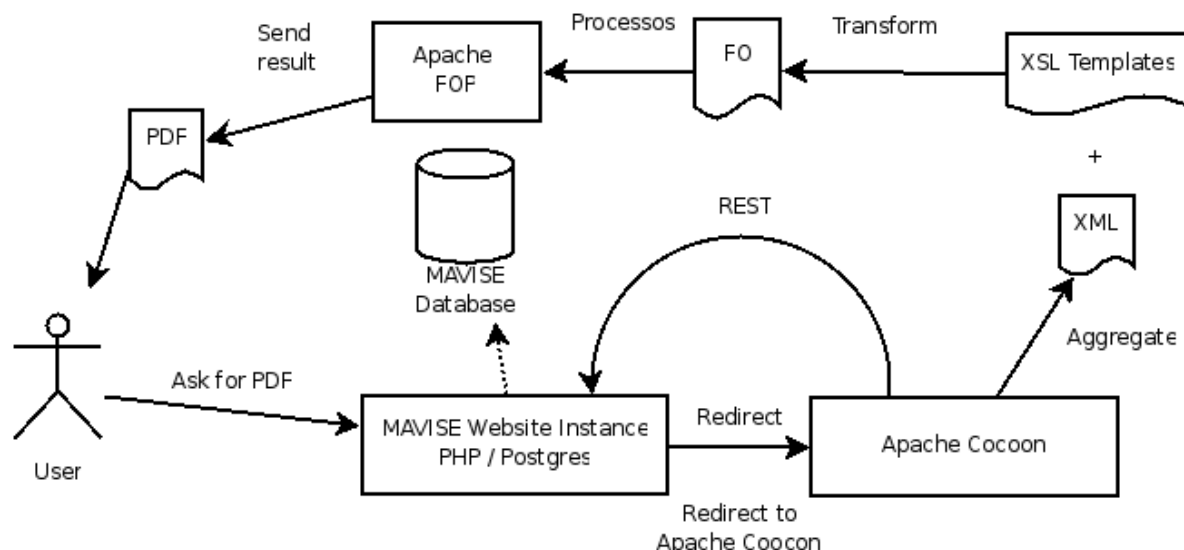
Alors à quoi bon « coder » à nouveau la présentation des documents PDF ?

Nous avons donc proposé au client de bâtir le système d'impression des documents PDF sur un système de template.

Ce système repose sur le langage de programmation [XSL-FO](#), les projets Apache [FOP](#) et Apache [Cocoon](#).

Architecture du système d'impression

<!-- htmlA -->



<!-- htmlB -->

1. Lorsque l'utilisateur clique sur le lien PDF, il est redirigé vers Apache Cocoon ;
2. Cocoon récupère l'ensemble des informations directement sur le site Web via l'API [REST](#) ;
3. Il agrège les informations dans un document XML ;
4. Ce document est transformé en FO à l'aide d'une feuille de style XSLT ;
5. Le document est produit par Apache FOP ;
6. L'utilisateur obtient le document final.

L'interface entre le Site Web et Apache Cocoon

L'architecture du système d'impression repose sur le fait que l'ensemble des variables manipulées par notre application est accessible à l'aide d'URL REST.

En effet, il est possible d'afficher le contenu des différentes variables manipulées par l'application à l'aide d'adresses URL spécifiques. Voici un exemple de sortie :

```
<?xml version="1.0" encoding="UTF-8"?>

<is_admin />

1
FR
France
t
<minimal_age_of_audience>4+</minimal_age_of_audience>

...
```

Il est donc possible de récupérer l'ensemble des informations du site à l'aide du protocole REST.

Nous avons aussi de très nombreux tableaux dans l'application. Pour les récupérer, nous utilisons le format XML natif du contrôleur de tableaux [dhtmlxGrid](#). Voici un exemple de sortie : [fichier XML pour dhtmlxGrid](#).

Agrégation des données dans un seul fichier

Pour récupérer l'ensemble des données du site dans un fichier XML, la directive « include » offerte par Cocoon a été utilisée :

```
$ cat /var/lib/tomcat5/webapps/cocoon/mavise/program/program.xml
<?xml version="1.0"?>

<i:include src='cocoon:/webui_program' />
...

```

Cette directive dit à Cocoon, d'inclure le contenu qui se trouve à l'url cocoon :/webui_program, défini dans le fichier sitemap.xmap :

```
<!-- Webui -->
<map:match pattern="webui_*/*">
<map:generate src='http://localhost/{1}?id={2}&presenter=rest&filter=webui' />
<map:serialize type="exml" />
</map:match>

```

Transformation XSLT

Pour transformer les données contenues dans ce fichier, nous avons créé notre propre feuille de style XSLT, pour produire un document FO. Cette feuille de style prend en compte la mise en forme des différentes données du site.

Grâce à ce système, l'ensemble de la mise en forme est centralisé dans un seul fichier.

Il est également possible d'ajouter du contenu statique dans le PDF à l'aide de balises dédiées, comme par exemple une balise copyright, qui sera transformée en un texte statique dans tous les documents PDF.

Le pipeline pour les documents PDF

Voici le pipeline final pour produire des documents PDF :

```
<!-- PDF -->
<map:match pattern="*/*/*.pdf">
  <map:generate src='{1}/{3}.xml' />
  <map:transform src="mk_id.xsl" type="xslt">
    <map:parameter name="val" value="{2}" />
  </map:transform>
  <map:transform type="include">
    <map:parameter name="parallel" value="true" />
    <map:parameter name="support-caching" value="true" />
    <map:parameter name="expires" value="600" />
  </map:transform>
```

Le point intéressant à noter dans ce pipeline est l'encodage des arguments passés via l'URL pour le document PDF. En effet, les URL pour produire les PDF sont formés de la manière suivante :

<http://mavise.obs.coe.int/cocoon/ma...>

L'URL contient le nom du module et l'identifiant de la chaîne. Il est ainsi possible de passer d'autres arguments au système d'impression (comme par exemple changer l'orientation du document).

Performances

Au niveau des performances du système d'impression, nous n'avons pas une charge très importante sur la génération des documents PDF. Mais la production des documents est une tâche complexe et nécessite donc de manière générale beaucoup de ressources.

Une chose intéressante à noter : la génération des documents PDF est plus rapide que l'affichage dans un navigateur ! Cela est lié au fait que les échanges de données se font en local sur le serveur, il y a donc moins de bande passante utilisée que dans le cas de l'affichage de cette même page par le navigateur.

Les petits désagréments du format PDF

Nous avons rencontré des difficultés avec la gestion des polices UTF-8. En effet, comme la base contient des données sur des chaînes de télévision et des entreprises Turques, nous avons eu besoin d'embarquer une police UTF-8 à l'intérieur du document.

Et visiblement cela pose des problèmes à certains lecteurs.

Quelques liens utiles

Voici quelques liens utiles qui nous ont aidé dans la réalisation de ce système d'impression :

- [Le guide d'installation d'Apache Cocoon, FOP sur Debian GNU/Linux](#) ;
- [Documentation de la version 2.1 d'Apache Cocoon](#) ;
- [XPath Tutorial](#) ;
- [XSLT Tutorial](#) ;
- [XSL-FO](#)