Collaboration de Zope et Mozilla – générer XUL à l'aide de Zope

Tomasz Rybak

Dans cet article, nous décrirons comment modifier le contenu émis à l'aide d'un serveur d'application Zope sur l'exemple d'une émission de HTML et XUL.

Sur l'auteur :

Chercheur au département d'Informatique de l'école polytechnique de Białystok, vous pouvez contacter l'auteur : *bogomips@post.pl*

La plupart des environnements disponibles actuellement, permettant de présenter le contenu sur Internet, sont capables d'utiliser uniquement le format HTML. À l'époque des navigateurs (aussi bien des programmes que des périphériques), capables d'afficher les fichiers de toute sorte, tels que XHTML, SVG et WML, c'est une restriction qui freine le développement de l'Internet. La solution à problème ne devrait pas consister à créer plusieurs versions du système pour des types de périphériques différents ni à forcer l'utilisateur de choisir le format de données qu'il veut recevoir.

XUL

XUL (en anglais *XML User-Interface Language*) est un langage qui sert à décrire l'interface utilisateur. Il est utilisé dans tous les programmes, basés sur le moteur Gecko, donc sur le paquet entier Mozilla ainsi que sur Firefox, Thunderbird, Sunbird, nvu et d'autres.

Il est possible d'afficher les fichiers XUL dans un navigateur, basé sur le moteur Gecko au moyen de plusieurs méthodes disponibles. La première d'entre elles consiste à télécharger le fichier en local depuis le disque. Pour ce faire, transmettez le nom du fichier au navigateur soit dans la ligne de commande lors du démarrage soit à l'aide de l'option du menu `File->Open`. La deuxième méthode consiste à envoyer le fichier XUL via le serveur de sorte que le fichier envoyé soit reconnu par Gecko comme une description ; il est nécessaire de paramétrer l'en-tête Content-Type sur `application/vnd.mozilla.xul+xml`. La troisième méthode consiste à installer l'extension (plug-in) dans le navigateur et à appeler les fichiers ainsi installés à l'aide du lien URL qui commence par chrome://. Il existe aussi une quatrième méthode qui ne nécessite pas le navigateur mais uniquement le moteur Gecko. Il existe un projet XUL Runner, permettant d'afficher les fichiers XUL et de

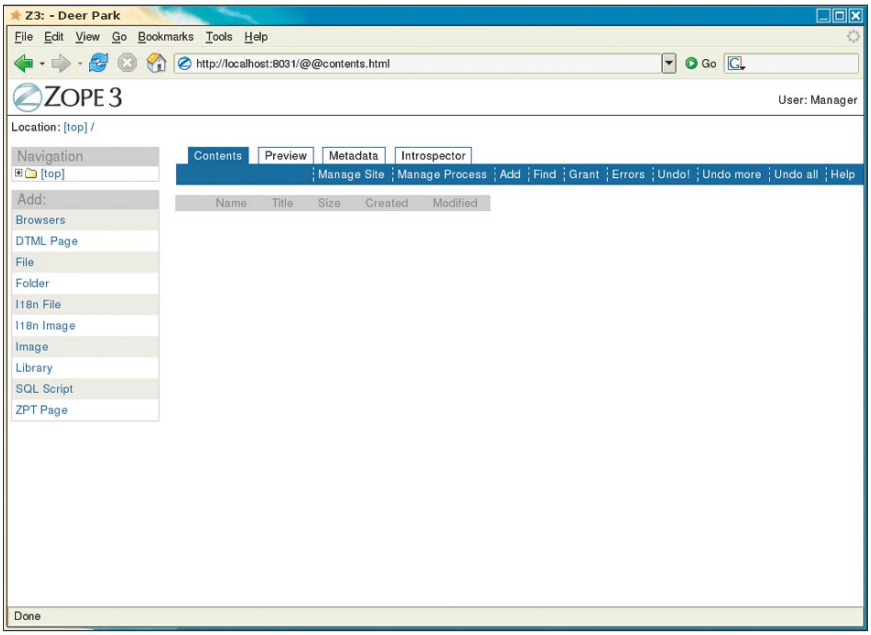

Figure 1. *Vue d'un exemple de l'écran de Zope dans le navigateur Mozilla Firefox*

lancer les modules XPCOM sans utiliser le navigateur. XUL Runner est un moteur Gecko, dépourvu des modules utilitaires (tels que navigateur, client de messagerie, etc.).

Les éléments de XUL se trouvent dans l'espace de noms *http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul*. L'élément principal parent de tous les autres éléments, est l'élément window. Il sert de description de la fenêtre, où se trouveront d'autres éléments et il sera affiché par le navigateur. Les fenêtres, de même que les autres éléments, peuvent avoir leurs propres identifiants (attribut `id`), uniques pour l'ensemble du fichier.

Présenter les données à l'aide de XUL

Les données, pourvues d'une structure interne, s'affichent en général au moyen des tables. La manière la plus souvent utilisée pour afficher les tables consiste à utiliser l'élément `tree`. Il est doté de deux sous-éléments. Le premier d'entre eux s'appelle `treecols` et décrit les colonnes de tables ; le second s'appelle `treechildren` et contient toutes les données affichées dans la table. Les éléments `treecol` sont des éléments enfants de l'élément `treecols` ; chacun d'entre eux devrait contenir un attribut `id` qui sert à identifier les colonnes lors du chargement de données et leur affichage ainsi qu'un attribut `label`, contenant une entrée, affichée dans l'en-tête de la colonne. L'élément `treechildren` contient les éléments `treeitem`, représentant les données. Tout élément `treeitem` devrait être doté d'un élément enfant `treerow`. Les éléments `treecell` se trouvent dans `treerow` en tant qu'éléments fils. Chaque élément `treecell` devrait contenir un élément `label` où sont enregistrées les données affichées dans cette colonne. Si le nombre d'éléments `treecell` est plus petit que celui de colonnes dans la table, les colonnes sont remplies des valeurs de gauche à droite. Si vous voulez forcer un autre ordre pour remplir les colonnes, vous pouvez utiliser l'attribut `ref` qui indique les colonnes (identifiées par l'attribut `id`) auxquelles appartiennent les valeurs déterminées.

Si la taille de la table n'est pas limitée par les éléments parents, elle dépend par défaut du nombre des lignes qu'elle contient. Si vous voulez que l'arbre ait une taille minimale, indépendamment du volume de données, paramétrez la valeur de l'attribut `rows` sur le nombre qui indique le nombre de lignes à afficher.

Comme son nom l'indique, l'élément `tree` peut également servir à afficher les données d'un arbre, autrement dit, les données dotées des données inférieures. Afin de les présenter et de pouvoir activer et désactiver l'affichage des données inférieures, l'une des colonnes doit avoir un attribut `primary` dont la valeur est `true`. L'icône, qui sert à dérouler et à masquer les sous-arbres, s'affichera dans cette colonne. De plus, il serait impossible de désactiver l'affichage de cette colonne.

Afin d'enregistrer les données de l'arbre (hiérarchiques), utilisez les mêmes éléments que pour les données tabulaires. Si l'une des lignes contient des données inférieures, l'élément `treeitem` devrait alors avoir l'attribut `container` dont la valeur est `true`. L'élément `treechildren` se trouve après l'élément `treerow` qui détermine les éléments à afficher dans la ligne du premier rangé ; l'élément `treechildren` décrit le sous-arbre adéquat.

Combiner XUL et HTML

Il est possible d'incorporer le contenu HTML dans le fichier XUL. Pour ce faire, il faut ajouter l'espace de noms HTML *http://www.w3.org/1999/xhtml* au fichier XUL ; cet espace est en général décrit à l'aide du préfixe *html:* ; il faut ensuite précéder les éléments HTML de ce préfixe. Les balises saisies doivent être conformes à XML ; les noms des éléments doivent être écrits avec des minuscules et les valeurs de tous les attributs doivent se trouver

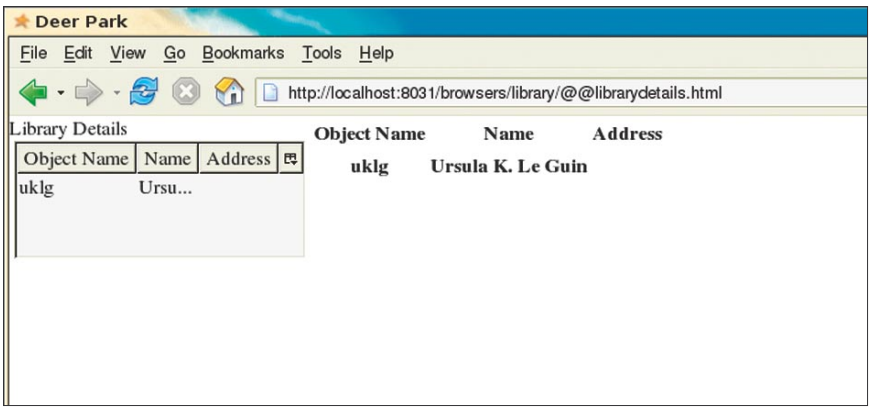

Figure 2. *Modèle XUL affiché dans le navigateur Mozilla Firefox*

entre les guillemets ou les apostrophes et chaque élément doit être terminé. Les mêmes règles sont en vigueur lorsqu'on écrit les pages en XHTML ; si vous connaissez donc XHTML, vous n'aurez aucun problème.

Zope

Zope est un serveur d'application, écrit en Python. Grâce à Python, ni Zope ni le code utilisé n'a aucun problème pour être transportable.

La version la plus récente de Zope porte le numéro 3.1.0. Les versions 3.x sont incompatibles avec Zope, version .x. Pour cette raison, Zope, version 3.x, est indiqué comme Zope X3 où X signifie une rétro-incompatibilité. Les concepteurs promettent toutefois de recommencer le travail sur le retour de la compatibilité pour que les anciens programmes puissent être exécutés dans la nouvelle version de Zope.

Zope est un système de composants, ce qui permet d'utiliser facilement le code plusieurs fois et de combiner des classes différentes. Le programmeur est chargé d'écrire des composants qui jouent des rôles différents et de décrire comment le serveur doit s'en servir et de les combiner. Les concepteurs de Zope recommandent d'opter pour des composants très simples et de ne pas écrire des composants qui jouent plusieurs rôles. Parmi de nombreux types de composants, nous utiliserons dans cet article les composants suivants :

- composants de contenu (en anglais *content*) ne sont en général dotés que des champs, dépourvus de méthodes et stockent les données affichées sur les pages,
- composants d'adaptation (en anglais *adapters*) servent à combiner plusieurs types d'autres composants et à adapter les composants à jouer des rôles différents,
- composants de vues (en anglais *views*) permettent de modifier la méthode par défaut et d'afficher les composants de contenu. Les composants de vues sont un type spécial d'adaptateurs.

Grâce à la structure de composants, Zope est capable de supporter de nombreux types de transmissions. Les serveurs HTTP, FTP et XML-RPC sont disponibles par défaut. Pour accéder aux données stockées dans les objets, il faut utiliser chacun de ces protocoles, bien qu'il soit parfois nécessaire d'écrire un adaptateur ; la structure de composants garantit toutefois qu'aucune modification ne soit nécessaire.

De même que la plupart des systèmes modernes, Zope se sert d'interfaces comme une description des contrats à remplir. Vous appelez les objets en tant qu'interfaces. Les fichiers de configuration sont le seul endroit où l'on parle de classes ; les classes y sont décrites comme des fabriques produisant les objets à implémenter dans les interfaces. Python n'offre pas d'interfaces mais elles sont disponibles en Zope, dans la classe `zope.interfaces.Interface`. Toute interface doit hériter de cette classe. Afin d'indiquer que la classe implémente une interface donnée, il faut utiliser la méthode `zope.interface.implements` ; son appel doit être placé en premier, immédiatement après la ligne contenant le nom de la classe (et après le commentaire qui décrit la classe). Cette méthode prend les arguments suivants : noms des interfaces implémentées par la classe donnée. La classe `Interface` fournit plusieurs méthodes. La méthode `providedBy` répond à la question si la classe implémente l'interface donnée. La méthode `implementedBy` retourne la liste de toutes les interfaces implémentées par la classe.

L'objectif principal des interfaces consiste à stocker les données. Pour

Listing 1. *Afficher les données hiérarchiques à l'aide de XUL*

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<!DOCTYPE window>
<window title="TreeView"
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
      <tree flex="1">
      <treecols>
         <treecol id="nom" label="Nom" flex="1"/>
         <treecol id="valeur" label="Waleur" primary="true"  flex="1"/>
      </treecols>
      <treechildren>
         <treeitem container="true">
            <treerow>
               <treecell label="A1"/>
               <treecell label="11"/>
            </treerow>
      <treechildren>
         <treeitem>
            <treerow>
               <treecell label="A2"/>
            </treerow>
         </treeitem>
      </treechildren>
         </treeitem>
      </treechildren>
      </tree>
</window>
```

Listing 2. *Combiner HTML et XUL*

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<!DOCTYPE window>
<window xmlns:html="http://www.w3.org/1999/xhtml"
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<vbox>
   <html:div>
      <html:a href="http://localhost/">Ten komputer</html:a>
      <html:br/>
   </html:div>
   <button label="Boutton XUL"/>
</vbox>
</window>
```

Listing 3. *Interfaces qui décrivent les composants implémentés*

```
#!/usr/bin/python
from zope.interface import Interface, Attribute, classImplements
from zope.schema import TextLine, Text, Int, Field
from zope.app.container.constraints import ContainerTypesConstraint
from zope.app.container.constraints import ItemTypePrecondition
from zope.app.container.interfaces import IContained, IContainer
class ILibrary(Interface):
   name = TextLine(title=u"Name of library",
      description=u"Contains name of library.",
      required=True)
   description = Text(title=u"Description of library",
      description=u"Human-readable description of library.",
      default=u"",
      required=False)
class IAuthor(Interface):
   name = TextLine(title=u"Name of author",
      description=u"Contains name of author.",
      required=True)
   address = Text(title=u"Address of author",
      description=u"Contains address of author.",
      default=u"",
      required=False)
class IBook(Interface):
   title = TextLine(title=u"Title of book",
      description=u"Contains title of book.",
      required=True)
   year = Int(title=u"Year of publication",
      description=u"Contains year of publication of book.",
      required=True, min=0)
   description = Text(title=u"Description of book",
      description=u"Contains description of book.",
      default=u"",
      required=False)
class IBookContained(IContained):
   __parent__ = Field(constraint = ContainerTypesConstraint(IAuthor))
class IAuthorContained(IContained):
   __parent__ = Field(constraint = ContainerTypesConstraint(ILibrary))
class IAuthorContainer(IContainer):
   def __setitem__(name, object):
      """Add an IBook."""
   __setitem__.precondition = ItemTypePrecondition(IBook)
class ILibraryContainer(IContainer):
   def __setitem__(name, object):
      """Add an IAuthor."""
   __setitem__.precondition = ItemTypePrecondition(IAuthor)
```

Listing 4. *Implémentation de la classe Author*

```
#!/usr/bin/python
from zope.interface import implements
from zope.app.container.btree import BTreeContainer
from interfaces import IAuthor
from interfaces import IAuthorContained
from interfaces import IAuthorContainer
class Author(BTreeContainer):
   implements(IAuthor, IAuthorContained, IAuthorContainer)
   def __init__(self, name = u"", address = u""):
      super(Author, self).__init__()
      self.name = name
      self.address = address
```

cette raison, nous rencontrons souvent les interfaces, dotées uniquement des champs, quoique cela puisse sembler non conformes à ce que vous avez déjà vu, lors du travail avec d'autres langages de programmation. Le nom et le type décrivent ces champs. En fonction du type, ils peuvent contenir des données différentes, affichées de manière différente. Les types utilisés le plus souvent sont :

- `Text` et `TextLine` qui servent à stocker le texte. `min_length` et `max_length` peuvent être les paramètres du constructeur de la classe de ce type.
- `Bool`, `Int`, `Float` sont les équivalents des types de Python. Ils peuvent (mis à part Bool bien évidemment) préciser les paramètres `min` et `max` dans le constructeur,
- `Tuple`, `List`, `Dict` sont les équivalents des collections de Python. Ils permettent d'afficher plusieurs objets. Les paramètres du constructeur : `min_length`, max_length et `value_type` qui déterminent le type stocké dans la collection. Le constructeur du dictionnaire est doté d'un paramètre supplémentaire : `key_type`, permettant de déterminer le type de clé.

Mis à part les paramètres susmentionnés, les constructeurs de toutes les classes peuvent avoir les paramètres suivants :

- `title` contient le nom affiché du champ,
- `description` contient la description affichée du champ donné,
- `required` prend la valeur logique qui indique si le champ donné doit être rempli lors de la création de l'objet,
- `readonly` détermine si la valeur du champ peut être modifiée après la création de l'objet,
- `default` permet d'ajouter une valeur par défaut,
- `order` permet de choisir un autre ordre d'affichage de champs que celui défini par la déclaration de l'objet.

D'après les types de champs, Zope génère un code HTML adéquat ; vous n'avez donc pas besoin de l'écrire vous-mêmes. Deux objets sont définis pour chaque classe ; l'un d'entre eux sert à afficher les données, le second – à télécharger les données du type approprié.

La Figure 1 présente l'écran de Zope. En haut à gauche, vous voyez la structure de l'arbre des objets qui existent dans l'instance donnée du serveur ; il est possible de dérouler et de masquer chaque branche. En dessous se trouve la liste d'objets à placer dans l'arbre d'objets. Si vous cliquez sur le nom, vous serez transportés à la page où vous écrivez les informations, indispensables à créer un objet. La partie principale de l'écran, en haut, présente une liste, permettant de choisir la manière d'afficher l'objet et les informations ; l'onglet Introspector permet de regarder la description du code, chargé d'implémenter l'objet donné, y compris les classes, les classes parents, les interfaces implémentées, etc. Cela peut être particulièrement utile lorsque vous apprenez à programmer et vous cherchez les erreurs car il présente comment Zope voit le code. Une liste des commandes se trouve en dessous ; il s'agit des commandes qu'il est possible d'effectuer sur l'objet donné. En dessous, dans la partie principale de l'écran s'affiche l'objet à proprement parler.

Description des données stockées

Afin de présenter les fonctionnalités du système, nous avons créé des interfaces `ILibrary`, `IAuthor` et `IBook`. Chaque objet de la bibliothèque contient des objets d'auteurs dont les ouvrages sont disponibles dans la bibliothèque et chaque objet de l'auteur contient les objets d'ouvrages écrits par l'auteur en question. Il est impossible qu'un ouvrage ait plusieurs auteurs et qu'un auteur appartienne à plusieurs bibliothèques. Il est cependant possible de créer un objet, représentant le même auteur dans chaque bibliothèque mais cela signifie que plusieurs objets de ce type existeront. Afin de forcer le stockage des objets d'ouvrages uniquement dans les objets d'auteurs et les objets d'auteurs uniquement dans les objets de bibliothèque, nous avons créé des interfaces qui héritent de `IContained` et `IContainer`. Ces interfaces utilisent le champ `__parent__` et la méthode `__setItem__` ; ils limitent les types d'objets qui peuvent être des objets parents et fils pour les types déterminés. Vous pouvez remarquer que le menu, permettant de créer de nouveaux objets, change en fonction du composant visualisé. De plus, si l'objet actuel est une bibliothèque, l'auteur est le seul type de l'objet.

Le code devrait se trouver dans le nouveau paquet dans le répertoire avec les fichiers de l'instance Zope. Dans le cas de l'installation à partir des sources, ce répertoire s'appelle */usr/local/Zope3-3.1.0/lib/python*, dans le cas de Debian, c'est */var/lib/zope3/instances/nazwainstancji/lib/python*. Dans ce répertoire, créez le répertoire, appelé *library,* où vous placerez les fichiers sources.

Afin de définir la classe qui implémente l'interface, vous pouvez appeler le programme `pyskel.py` à qui vous précisez en paramètre le nom complet de l'interface. Le script crée un squelette de la classe, en écrivant toutes les méthodes indispensables ; vous ne devez remplir que les corps de ces méthodes et ajouter la documentation.

Si vous voulez stocker d'autres objets dans l'objet (comme dans l'exemple des auteurs dans la bibliothèque et des livres dans les auteurs), il vous faut implémenter les containers. `BTreeContainer` est le container le plus souvent utilisé. Comme son nom l'indique, il est un arbre B. Le Listing 4 présente l'implémentation de la classe `Author` (les autres classes la ressemblent).

Configuration

Une fois le code de composants créé, il faut informer Zope quels composants sont disponibles et comment communiquer avec eux. La configuration de Zope se déroule à l'aide des fichiers ZCML (en anglais *Zope Configuration Markup Language*), qui sont les fichiers XML. *http://namespaces.zope.org/zope* est l'espace de noms, utilisé dans les fichiers de configuration. Le fichier principal de configuration contient en général les informations concernant les interfaces et les classes de ce paquet.

Listing 5. *Configuration des composants, chargés de stocker les informations sur les auteurs*

```
<interface
   interface=".interfaces.IAuthor"
   type="zope.app.content.interfaces.IContentType"/>
<content class=".author.Author">
   <implements
      interface="zope.app.annotation.interfaces.IAttributeAnnotatable"/>
   <implements
      interface="zope.app.container.interfaces.IContentContainer"/>
   <factory
      id="library.author.Author"
      description="Author"/>
   <require
      permission="zope.ManageContent"
      interface=".interfaces.IAuthor"/>
   <require
      permission="zope.ManageContent"
      interface=".interfaces.IAuthorContainer"/>
   <require
      permission="zope.ManageContent"
      set_schema=".interfaces.IAuthor"/>
</content>
```

Listing 6. *Configuration d'affichage de données sur les auteurs*

```
<addform
   label="Add Author"
   name="AddAuthor.html"
   schema="library.interfaces.IAut
                        hor"
   content_factory="library.author.
                        Author"
   fields="name address"
   permission="zope.ManageContent"/>
<addMenuItem
   class="library.author.Author"
   title="Author"
   description="An Author"
   permission="zope.ManageContent"
   view="AddAuthor.html"/>
<editform
   schema="library.interfaces.IAut
                        hor"
   for="library.interfaces.IAuthor"
   label="Change Author"
   name="edit.html"
   permission="zope.ManageContent"
   menu="zmi_views" title="Edit"/>
<containerViews
   for="library.interfaces.IAuthor"
   index="zope.View"
   contents="zope.View"
   add="zope.ManageContent"/>
```

Le fichier de configuration du paquet entier s'appelle *configure.zcml* et se trouve dans le répertoire du paquet ; le répertoire de configuration ne contient que le fichier avec les informations sur votre fichier de configuration. Ce fichier se présente en général de manière suivante : `<include package=''nom_paquet''/>`.

`configure` est l'élément principal du fichier de configuration. Parmi ses sous-éléments se trouvent `interface`, dont les attributs sont les suivants : l'interface contenant le nom complet de l'interface décrite. Un autre sous-élément s'appelle `type` et il indique le composant de l'interface. Puisque nous décrivons le composant du contenu, nous écrivons `zope.app.content.interfaces.IContentType` dans la valeur de cet attribut. Si le nom de l'interface ou de la classe commence par un point, il s'agit d'un nom relatif et non absolu ; Zope ajoute alors le nom du paquet décrit avant le point.

L'élément `content` décrit la classe qui est un composant. Son attribut `class` indique le nom de la classe. Il est doté de plusieurs sous-éléments. Les éléments `implements` décrivent les interfaces implémentées par la classe donnée ; une instance de cet élément décrit une interface. L'élément `factory` décrit la classe chargée de créer une classe donnée. Le modèle de fabrique permet de séparer la classe et la méthode de sa création. Les éléments `require` dans l'attribut `permission` décrivent les droits, indispensables pour utiliser l'interface, indiquée dans l'attribut `interface`.

Le Listing 5 présente un exemple de configuration (fragment du fichier de configuration concernant l'interface `IAuthor` et la classe Author).

Listing 7. *Page modèle en HTML*

```
<html
   xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Library details</title>
</head>
<body>
   <h1>Library Details</h1>
   <table>
   <div tal:replace="structure view/authors" />
   </table>
</body>
</html>
```

Listing 8. *Classe du modèle*

```
#!/usr/bin/python
from zope.app import zapi
from zope.app.publisher.browser import BrowserView
from zope.app.pagetemplate.viewpagetemplatefile import ViewPageTemplateFile
from library.interfaces import ILibrary
from library.interfaces import IAuthor
class LibraryView(object):
   def __init__(self, context, request):
      self.context = context
      self.request = request
   def listAuthors(self):
      children = []
      for name, child in self.context.items():
         if IAuthor.providedBy(child):
            info = {}
            info['objectname'] = name
            info['name'] = child.name
            info['address'] = child.address
            children.append(info)
      return children
   authors = ViewPageTemplateFile("libraryauthors.html.pt")
class LibraryViewHTML(LibraryView):
   def __init__(self, context, request):
      super(LibraryViewHTML, self).__init__(context, request)
   authors = ViewPageTemplateFile("libraryauthors.html.pt")
class LibraryViewXUL(LibraryView):
   def __init__(self, context, request):
      super(LibraryViewXUL, self).__init__(context, request)
      response = self.request.response
      response.setHeader("Content-Type", "application/vnd.mozilla.xul+xml")
   authors = ViewPageTemplateFile("libraryauthors.xul0.pt")
   authors_html = ViewPageTemplateFile("libraryauthors.xul1.pt")
```

Afficher le contenu

Le paquet browser est chargé d'afficher les objets ; c'est en général un paquet fils du paquet principal.

Comme nous l'avons déjà mentionné, le code HTML est généré d'après les types de champs, contenus dans les interfaces.

La configuration de l'affichage est décrite dans le fichier *configure.zcml* du paquet `browser`. Les éléments, responsables de la configuration de l'affichage, se trouvent dans l'espace de nom *http://namespaces.zope.org/browser*. De même que le fichier de configuration du paquet entier, `configure` est un élément principal. Ses éléments enfants permettent d'enregistrer la manière d'afficher tous les composants. L'élément `addMenuItem` est chargé d'ajouter la position au menu, permettant de créer de nouveaux objets. Cette position permet de créer un objet indiqué par l'attribut `class`. La position du menu sera décrite à l'aide de la valeur `title`. L'attribut `permission` indique le niveau de droits nécessaires pour créer un objet donné. L'attribut `view` indique le nom de la page utilisée pour télécharger les données indispensables pour créer cet objet. Si vous cliquez sur une position du menu, le navigateur sera redirigé à la page indiquée par cet attribut.

L'élément `addform` est chargé de récupérer les données indispensables pour créer l'objet dont l'interface est indiquée à l'aide de l'attribut `schema`. Sa classe est en revanche chargée de créer des objets qui implémentent cette interface via `content_factory`. Le titre de la page se trouve dans l'attribut `label` et le nom du fichier HTML généré se trouve dans l'attribut `name`. L'attribut `fields` détermine les valeurs de champs à télécharger chez l'utilisateur.

L'élément `editform` décrit la page permettant de modifier les valeurs des données stockées dans l'objet. L'attribut `menu` indique le menu où doit se trouver le lien vers la page permettant une édition. La description de la position du menu est récupérée dans l'attribut `title`. L'élément `containerViews` permet de présenter une liste d'objets fils, stockés dans l'objet donné, en tant qu'une des vues possibles.

Pour que Zope prenne en compte la configuration contenue dans le paquet browser, il faut ajouter l'information sur l'emplacement du fichier de configuration de l'affichage au fichier principal de configuration. Pour ce faire, ajoutez (en tant qu'un élément fils de l'élément `configure`) la ligne :

```
<include package=".browser"/>
```

Le Listing 6 présente un exemple de fichier qui configure l'affichage des auteurs.

Si vous ne voulez pas que Zope affiche les objets de manière standard, créez vos propres vues. Pour ce faire, vous avez besoin d'un fichier de modèle et, bien que cela ne soit pas toujours obligatoire, une classe de modèle. Le fichier de modèle est en général un fichier HTML qui contient la manière de charger les données, au lieu des données à proprement parler. Autrement dit, il s'agit des méthodes qui retournent les données à afficher. Lors de la génération du fichier HTML du fichier de modèle, Zope ajoute les données obtenues après avoir appelé les méthodes indiquées.

Vous indiquez les méthodes appelées à l'aide de l'attribut `attribut:content` dont la valeur est `classe/methode` où `methode` est le nom de la méthode appelée, retournant les données affichées. `Classe` prend en général soit la valeur `context`, ce qui signifie le chargement de données directement de l'objet affiché, soit la valeur `view`, ce qui indique le chargement de données via l'appel de la méthode de la classe de vue.

Grâce à l'attribut `tal:context`, la valeur retournée est placée en tant qu'un fils de l'élément où vous avez utilisé cet attribut ; la valeur courante de cet élément est supprimée. Si vous voulez modifier la valeur de l'attribut et non de l'élément, utilisez `attribut:attributes` ; cet attribut prend en valeur deux éléments séparés par un espace. Le premier élément est le nom de l'attribut dont vous modifiez la valeur et le second élément, de même que pour `attribut:content`, est une méthode que vous appelez pour obtenir la valeur.

Si la méthode utilisée retourne plusieurs valeurs, autrement dit, elle retourne une collection (liste, tableau, dictionnaire), il est possible de créer un élément répétitif dans le modèle. Pour ce faire, placez les éléments répétitifs à l'intérieur de l'élément qui contient l'attribut `attribut:repeat` dont la valeur comprend aussi deux éléments. Le deuxième élément (de même que les attributs précédents) est une méthode utilisée pour télécharger les données. L'identifiant d'un élément de la collection est un second élément. Lors de la génération du contenu, Zope attribuera les objets de la collection à la variable dont le nom est égal à cet identifiant. Il est possible

Listing 9. *Page modèle en XUL*

```
<window
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
   xmlns:html="http://www.w3.org/1999/xhtml">
<hbox>
<vbox>
   <label value="Library Details"/>
   <tree rows="3">
      <treecols>
      <treecol id="objectname" label="Object Name"/>
      <treecol id="name" label="Name"/>
      <treecol id="address" label="Address"/>
      </treecols>
      <treechildren tal:replace="structure view/authors"/>
   </tree>
</vbox>
<html:table tal:replace="structure view/authors_html"/>
</hbox>
</window>
```

Listing 10. *Configuration de nouveaux modèles*

```
<page
   name="librarydetails.html"
   for="library.interfaces.ILibrary"
   template="librarydetails.html.pt"
   class="library.browser.librarydetails.LibraryViewHTML"
   permission="zope.View"
   menu="zmi_views" title="Details"/>
<page
   name="librarydetails.xul"
   for="library.interfaces.ILibrary"
   class="library.browser.librarydetails.LibraryViewXUL"
   template="librarydetails.xul.pt"
   permission="zope.View"
   menu="zmi_views" title="XUL"/>
```

Listing 11. *Configuration de l'objet de navigation*

```
<factory id="zope.BrowserFolder"
   component=".browserFolderFactory"/>
<browser:addMenuItem factory="zope.BrowserFolder"
   title="Browsers"
   description="Folder for diffenent browsers"
   permission="zope.ManageContent"/>
<view
   for=".IBrowserFolder"
   type="zope.publisher.interfaces.browser.IBrowserRequest"
   factory=".BrowserTraverser"
   provides="zope.publisher.interfaces.browser.IBrowserPublisher"
   permission="zope.View"/>
```

Bibliographie

- Philipp von Weitershausen *Web Component Development with Zope 3* Springer-Verlag Berlin 2005,
- Stephan Richter *Zope 3 Developer's Handbook* 2005.

d'appeler cet objet à l'intérieur de l'élément répété du modèle (par exemple, en précisant son nom au lieu du mot `context` ou `view`). Si l'une des méthodes retourne le code HTML, que vous voulez placer dans le fichier généré, utilisez alors l'attribut `attribut:replace`.

Le modèle ainsi créé constituera une page autonome. Si vous voulez que Zope place votre modèle dans la fenêtre standard (donc avec tous les menus standard), placez l'attribut `metal:use-macro` dont la valeur est `views/standard_macros/view` dans l'élément principal du modèle. Cette démarche permettra d'ajouter les scripts metal, chargés de modifier la page et d'ajouter les composants standard Zope.

Si vous avez placé les liens sur les méthodes de la classe du modèle dans le fichier du modèle, il faut la définir. Il s'agit d'une simple classe de Python, qui hérite de la classe `object`. Son constructeur prend deux paramètres. Le paramètre `request` contient des informations sur la requête envoyée par l'utilisateur ; le paramètre `context` contient le lien à l'objet affiché au moment même.

Une fois définie, la vue doit être enregistrée dans le système à l'aide de l'entrée dans le fichier ZCML. Afin d'enregistrer la vue, utilisez l'élément `page`. L'attribut `name` décrit le nom du fichier qui sera placé dans l'URL ; ce nom indique l'utilisation de la vue. L'attribut `for` contient le nom complet de l'interface pour laquelle vous créez la vue. L'attribut `class` indique la classe qui implémente la vue (le lien vers cette classe sera transmis au constructeur en tant que paramètre `context`). L'attribut `template` indique le fichier avec le modèle. L'attribut `menu` montre le menu dans lequel il faut placer le lien à la vue (on y place en général `zmi_view`). L'attribut `title` montre en revanche la description de l'option, placée dans ce menu.

Les Listings 7 et 8 présentent l'exemple de vue HTML pour la classe qui décrit la bibliothèque. Comme vous pouvez le constater sur l'exemple, le modèle ne doit pas constituer un seul fichier. Il est possible d'y ajouter d'autres modèles à l'aide de la classe `zope.app.pagetemplate.viewpagetemplatefile.ViewPageTemplateFile`. C'est une classe standard, chargée de supporter les modèles. Son constructeur prend le nom du fichier du modèle en paramètre. Si vous créez le champ dans la classe du modèle, dont la valeur est l'objet de la classe `ViewPageTemplateFile`, et ensuite, vous appelez ce champ dans un autre fichier du modèle (à l'aide de l'attribut `attribut:replace`), le modèle sera placé dans un autre fichier, comme le démontre l'exemple.

Si vous affichez votre modèle, vous remarquerez qu'à la fin de l'URL se trouve le nom de la vue que vous avez précisé dans l'attribut `name`. Ce nom est précédé de deux caractères @. Zope indique les vues à l'aide de ce préfixe.

Listing 12. *Implémentation de l'objet de navigation*

```
#!/usr/bin/python
from os.path import exists
import re
from zope.component.interfaces import IFactory
from zope.app.folder import Folder
from zope.app.folder.interfaces import IFolder
from zope.interface import implements, implementedBy
from zope.interface import directlyProvides, directlyProvidedBy
from zope.publisher.interfaces import NotFound
from zope.app import zapi, servicenames
from zope.app.container.traversal import ContainerTraverser
from zope.app.traversing.namespace import namespaceLookup
class IBrowserFolder(IFolder):
   pass
class BrowserFolderFactory(object):
   implements(IFactory)
   def __call__(self):
      folder = Folder()
      directlyProvides(folder, directlyProvidedBy(folder),
         IBrowserFolder)
      return folder
   def getInterfaces(self):
      return implementedBy(Folder)+IBrowserFolder
class BrowserTraverser(ContainerTraverser):
   __used_for__ = IBrowserFolder

   def publishTraverse(self, request, name):
      can = True
      obj = super(BrowserTraverser, self).publishTraverse(request, name)
      traversalstack = request.getTraversalStack()
      originaltraversalstack = request.getTraversalStack()
      if re.search("^@@.*\.html$", traversalstack[-1]) and re.search("Gecko",
                     request.getHeader('User-Agent')) is not None:
         traversalstack[-1] = re.sub("\.html$", ".xul", traversalstack[-1])
         try:
            namespaceLookup('view', traversalstack[-1][2:], obj, request)
         except:
            can = False
         if can:
            request.setTraversalStack(traversalstack)
      return super(BrowserTraverser, self).publishTraverse(request, name)
browserFolderFactory = BrowserFolderFactory()
```

Sur Internet

- Site avec des liens aux sites qui décrivent XUL *http://www.mozilla.org/projects/xul/*
- Description formelle de XUL *http://www.mozilla.org/xpfe/xulref/XUL_Reference.html*
- Site contenant les descriptions des éléments, les guides et l'introduction à XUL *http://www.xulplanet.com/*
- Projet XUL Runner, permettant de regarder les fichiers XUL sans utiliser le navigateur *http://developer.mozilla.org/en/docs/XULRunner*
- Matériaux d'une rencontre SLUG, consacrée à XUL *ftp://slug.optics.polsl.gliwice.pl/gnupolsl/16-xul/*
- Site officiel du projet Zope *http://www.zope.org/*

Il n'est pas nécessaire que les vues soient uniquement des fichiers HTML. Le Listing 9 présente la vue, définie en XUL. Elle peut utiliser la même classe du modèle, en se distinguant uniquement par un fichier de modèle. Si vous affichez toutefois cette vue dans le navigateur, vous remarquerez que le fichier XUL généré ne s'est pas affiché correctement. C'est ainsi car Zope a envoyé `text/html` comme valeur de l'en-tête `Content-Type`. Il faut le modifier. Pour ce faire, créez une nouvelle classe du modèle (qui hériterait de la classe précédente, utilisée pour le modèle HTML) et paramétrez l'en-tête `Content-Type` sur la valeur `application/vnd.mozilla.xul+`xml dans son constructeur. Le Listing 10 présente la configuration finale des modèles.

Chemin de la requête

Vous avez réussi à définir deux vues différentes. Le choix de la vue est effectué par l'utilisateur qui l'indique à l'aide de l'URL, et non par le système, d'après ce qu'affiche le navigateur. Si vous regardez attentivement l'URL, indiquant la vue, vous remarquerez qu'il contient le chemin entier qui mène à l'objet affiché. L'objet de navigation (en anglais *traverser*) est chargé de changer l'URL en liens vers les objets et de trouver ces objets. Il est possible de créer votre propre objet de navigation et de l'utiliser à la place de l'objet standard, joint à Zope. Pour ce faire, vous devez créer votre propre objet du répertoire (en anglais *folder*) et attribuer votre objet à ce répertoire. La classe de navigation doit implémenter la méthode chargée de parcourir l'URL et de trouver tous les objets indiqués. Cette méthode s'appelle `publishTraverse` et prend deux paramètres. Le premier d'entre eux est le nom de l'objet actuel qu'il faut trouver et le second est une requête (en anglais *request*) envoyée par l'utilisateur. L'un des champs de la classe `request` est l'objet `response`, contenant la réponse envoyée à l'utilisateur. De plus, l'objet de la classe `request` contient le champ `traversal_stack`, que vous appelez via les méthodes `getTraversalStact` et `setTraversalStack`. La valeur de ce champ est une liste des parties de l'URL qui indiquent les noms des objets à charger par l'objet de navigation. Le Listing 12 présente l'implémentation de la classe du répertoire et de l'objet de navigation. Nous y vérifions si le client appelle la vue en analysant si la dernière partie de l'URL (chargée via la méthode `getTraversalStack`) commence par deux caractères @. Si c'est le cas, essayez alors de changer la vue de HTML en XUL et vérifiez si vous réussissez. Pour le vérifier, appelez la fonction `lookupNamespace` avec le paramètre égal au nom de la nouvelle vue. La fonction prend deux paramètres ; le second est le nom de l'espace de nom que vous parcourez. Vous utilisez l'espace `view`, de même que Zope qui commence à parcourir l'espace `view` s'il voit deux caractères @ dans l'URL. Si vos recherches réussissent, précisez la valeur modifiée de la liste et appelez la fonction parent. La modification de la vue envoyée dépend de la valeur de l'en-tête de la requête `User-Agent`. Vous ne modifiez la vue que si l'en-tête contient le nom *Gecko*.

La Figure 2 présente une vue sélectionnée automatiquement. Remarquez que malgré la modification de la vue de HTML en XUL, l'URL, qui indique le chemin de cette vue, n'a pas été modifié. Sa dernière partie indique toujours la vue HTML. La modification n'a été effectuée qu'à l'intérieur de l'objet de navigation.

Le défaut de cette méthode consiste en fait que votre propre objet n'est utilisé que dans les répertoires indiqués dans la configuration. Ces répertoires utilisent leurs propres objets de répertoires. Il est donc nécessaire de placer les objets dont vous voulez modifier automatiquement les vues dans vos propres répertoires. Cette caractéristique peut ne pas être considérée comme défaut car elle permet de diviser les objets en objets dont la vue est sélectionnée automatiquement et objets dont la vue est définie par l'utilisateur.

Conclusion

La manière décrite n'épuise pas les possibilités pour modifier les manières de présenter le contenu. Au lieu d'utiliser XUL, il est possible de transmettre des fichiers SVG ou WML aux clients. Il est également possible d'envoyer des fichiers PDF ou OpenOffice.org, ce qui nécessite toutefois d'utiliser des classes capables d'enregistrer les fichiers dans ces formats. Le choix de la manière de présenter le contenu ne doit pas non plus dépendre du contenu de l'en-tête `User-Agent`. Il est possible que les données présentées dépendent du pays (ou l'adresse IP) d'où vient la requête, des contenus acceptés (en-tête `Accept-Content`) ou des langues (en-tête `Accept-Language`). Il nous reste à espérer que les concepteurs des services Internet utiliseront pleinement les possibilités d'afficher les données, proposées par les navigateurs actuels.

Installation de Zope

La version 3.1.0 de Zope requiert la version 2.3.5 de Python. Si vous disposez d'une autre version, précisez le nom du fichier qui est l'interpréteur de Python dans la valeur du paramètre `--with-python` du script *configure*. Zope est installé par défaut dans le répertoire */usr/local/Zope-3.1.0* ; il est également possible de le modifier à l'aide du paramètre `--prefix`.

L'installation de Zope consiste à décompresser les sources et à exécuter trois commandes traditionnelles :

```
./configure && make && make install
```

Avant de faire `make install`, il est recommandé d'effectuer des tests à l'aide de `make check`. Cette commande vérifie si la compilation s'est déroulée correctement et elle effectue des tests de compatibilité. Après plusieurs minutes, vous devriez obtenir un résultat positif. Ensuite, vous pouvez installer les fichiers sur le disque.

Une fois les fichiers installés, il est nécessaire de créer une instance du serveur Zope, autrement dit, une base de données et une structure de répertoires où vous placerez le code écrit. Pour créer l'instance, appelez le script *bin/mkzopeinstance*, installé dans le répertoire où vous avez installé Zope. Ses paramètres sont les suivants : le nom et le mot de passe de l'utilisateur qui sera administrateur de l'instance donnée, ainsi que le répertoire où sera placée l'instance. Voici un exemple d'appel :

```
./bin/mkzopeinstance -u admin:admin -d
                              /var/Zope.
```

Une fois l'instance créée, lancez-la. Pour ce faire, allez dans son répertoire et appelez le script ./bin/runzope ; la console sera ainsi occupée et l'instance donnée de Zope pourra être désactivée en appuyant sur *[Ctrl-C]*. Puisqu'il est possible d'utiliser cette solution sur le serveur, Zope propose aussi un script zopectl, similaire au script *apachectl*.

Zope est également disponible dans les distributions ; en Debian, son paquet s'appelle *zope3*. L'installation consiste à faire une commande `apt-get install`

`zope3` (ou `aptitude install zope3` qui est recommandée). Le script *dzhandle* est installé avec Zope ; ce script permet d'installer et de gérer l'instance de Zope. La configuration se trouve dans le répertoire */etc/zope3/nom_instance* tandis que les fichiers de l'instance dans le répertoire */var/lib/zope3/instance/nom_instance*.

Les scripts de démarrage s'installent avec le paquet *zope3* ; ils se trouvent dans le répertoire */etc/init.d*, ce qui permet de lancer le serveur Zope au démarrage du système.

Il est recommandé d'installer le paquet *zope3-sandbox* à des fins de développement ; ce paquet créé une instance, destinée aux tests et à la programmation. ■