15

Un site d'entreprise : Internet, intranet, extranet

Afin de mettre en pratique les connaissances qui ont été acquises sur Zope, nous allons nous pencher sur un problème auquel on ne peut manquer d'être confronté sur le Web : la construction du site d'une entreprise de taille moyenne. Alors que dans le chapitre précédent nous avons donné quelques éléments de conception, nous allons maintenant aborder l'implémentation d'un tel site, en commençant par en définir les besoins.

Les besoins

L'entreprise que nous allons maintenant décrire est une fabrique d'emballages en carton, dont le site web s'appelle `Weballage`. Cette entreprise emploie une centaine de personnes, réparties sur trois sites géographiques et trois services principaux :

- services généraux (dont le service informatique) ;
- fabrication, conditionnement et expédition ;
- direction.

Le service informatique de l'entreprise souhaite rassembler différentes applications hétéroclites au sein d'une seule interface : l'Internet. Aussi, le site Internet de l'entreprise doit être vu comme une application capable de réaliser les tâches suivantes :

- présenter des informations sur l'entreprise (activités…) ;
- proposer aux salariés de l'entreprise un annuaire interne ;

- présenter des informations au jour le jour, qui doivent être accessibles soit aux clients, soit aux salariés, soit aux visiteurs « anonymes » du site.

Ces activités regroupent donc celles d'un site Internet, d'un intranet (annuaire d'entreprise et informations aux salariés) et d'un extranet (informations aux clients).

Nous allons décrire brièvement chaque domaine fonctionnel du site.

Informations sur l'entreprise

Cette partie devra être composée de pages statiques, qui seront accessibles à tout le monde, mais qui ne seront modifiées que par la direction. Les pages seront d'une mise en pages riche et pourront contenir des images ou des vidéos. Leur contenu ne subira que de rares modifications.

Actualités

Le site propose aussi la gestion d'informations au jour le jour, écrites par un responsable d'un des trois services de la société et accessibles, suivant la nature de la « dépêche », aux salariés uniquement, aux salariés et aux grands comptes, ou à tous les visiteurs du site. Il s'agit d'un texte court, doté d'un titre, d'une date, d'un contenu et d'un résumé, et dont l'intérêt diminue avec le temps (il conviendra donc de bien mettre en évidence, dans un coin de chaque page par exemple, les actualités les plus récentes).

Annuaire interne

Cette partie du site est remplie par les salariés eux-mêmes. Ils peuvent remplir les champs usuels : nom, prénom, site, ligne directe, adresse e-mail, y adjoindre éventuellement une photographie et quelques commentaires personnels libres (CV, coordonnées personnelles, hobbies…). Dans tous les cas, la mise en pages est figée, mais les commentaires personnels peuvent comporter quelques enrichissements (décomposition en titres, sous-titres, liens vers d'autres pages…).

Chaque salarié ne peut modifier que sa propre page, mais peut consulter toutes les autres. La direction a la possibilité de supprimer une page (en cas de départ d'un salarié, par exemple).

Analyse

Cette section présente une analyse rapide du site et nous amènera tout naturellement à décrire les solutions que Zope nous propose pour sa réalisation.

Nous n'avons pas ici la prétention de présenter une méthode d'analyse générique, ou basée sur une quelconque méthode industrielle : il s'agit simplement de présenter des problèmes courants et les solutions que Zope est susceptible d'y apporter.

Qui fait quoi ?

Au vu des différentes parties du site que nous venons d'identifier, nous pouvons dresser un tableau des besoins pour les différentes sections du site.

Section	Accès en lecture	Accès en écriture	Contenu
Informations sur l'entreprise	*Tout le monde (section publique)*	*Direction*	*Pages HTML (richesse de la mise en page).*
Annuaire interne	*Salariés*	*Direction* *Personne concernée*	*Champs de taille fixe et texte libre pour les commentaires personnels, moteur de recherche.*
Dépêches	*Soit salariés, soit salariés et clients, soit tout le monde*	*Responsable de service*	*Texte, généralement court, date et nom de l'auteur, liens éventuellement, pas d'images.*

Objets métier

Zope publie des objets ; le tableau précédent va nous permettre de préparer une ébauche des objets que nous allons publier dans le site :

- Les pages HTML de présentation de la société pourront être des `DTML Document`, permettant éventuellement à leur auteur d'inclure le contenu dynamique proposé par Zope.
- L'annuaire est constitué d'objets potentiellement catalogables : il s'agira dans notre cas de ZClasses, la conception d'un produit n'étant pas nécessaire ici.
- Il faudra également que l'on puisse rechercher les dépêches avec le ZCatalog : là encore, la ZClasse se prête bien à un tel exercice.
- Pour la gestion des utilisateurs, nous utiliserons le classique `acl_users` de Zope, qui correspond parfaitement aux contraintes qu'induit le problème. Nous pourrions créer des classes utilisateur intégrant la gestion de l'annuaire, mais cela impliquerait l'installation d'un mécanisme d'authentification des utilisateurs non standard. Nous considérerons qu'un *utilisateur d'annuaire* et un *utilisateur du site* sont deux objets distincts ; cela est vrai, en particulier, pour les clients qui ne sont pas répertoriés dans l'annuaire.

Conception graphique du site

Un site web doit respecter une charte graphique, uniforme au fil des pages. Le DTML et la construction dynamique de pages par le serveur vont nous permettre de conserver la même présentation à travers tout le site, tout en évitant aux auteurs des pages de devoir maîtriser HTML pour intégrer ces pages.

L'essentiel de la conception graphique consiste à réaliser (ou, mieux, faire réaliser par un graphiste) une page « témoin » du site, c'est-à-dire servant de modèle au reste du site (voir figure 15-1). Pour cela, n'importe quel outil de conception de pages web peut être utilisé : le

seul objectif en l'occurrence est d'obtenir une page représentative du site, contenant suffisamment d'informations pour bâtir la structure physique dans Zope.

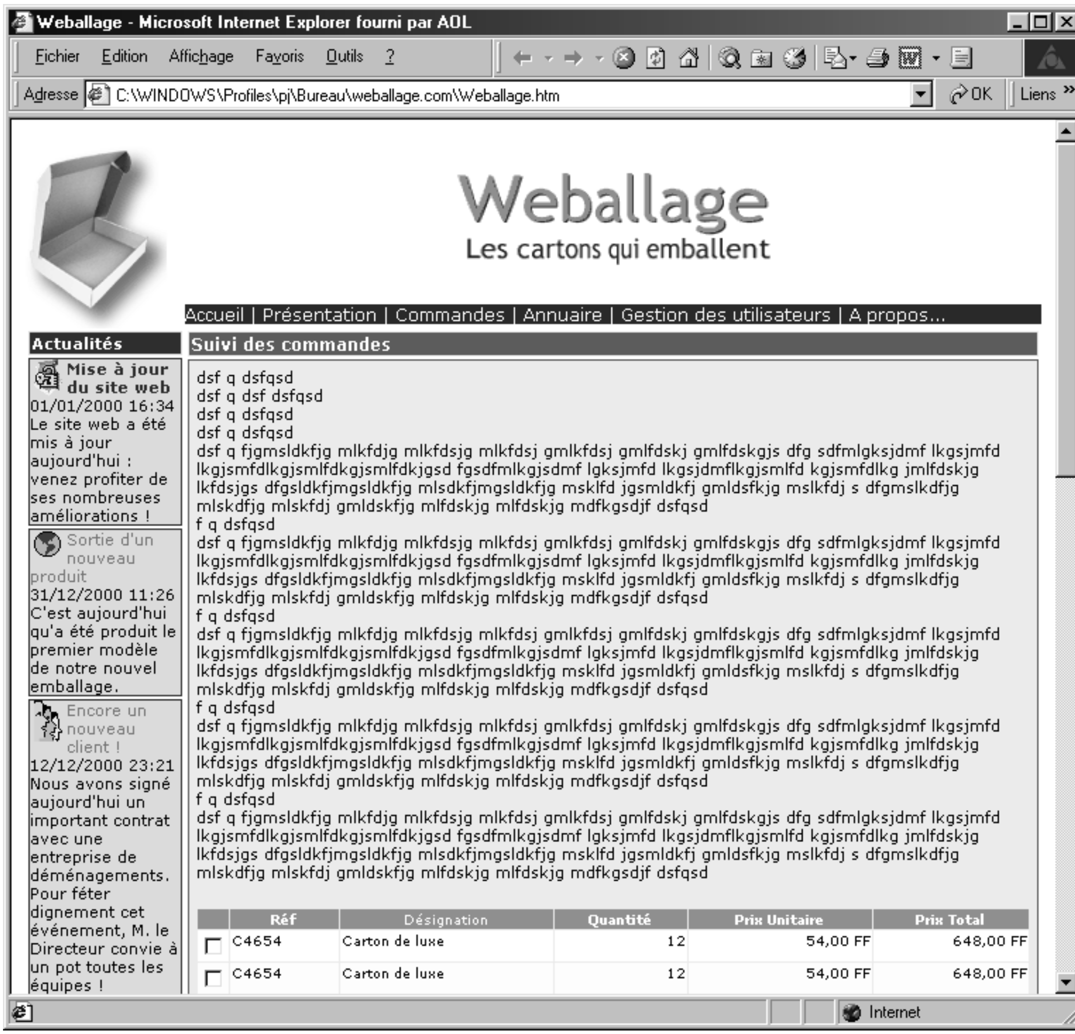


Figure 15-1

Page témoin

L'aspect définitif du site pourra différer de cette page, mais nous veillerons, lors de l'intégration de la charte graphique sous Zope, à bien respecter cette page témoin.

Architecture du site

Pour construire ce site, nous allons nous baser sur l'architecture présentée au chapitre 14, et l'étendre pour répondre aux besoins de ce site.

Architecture de présentation

Avant de se pencher sur le code HTML proprement dit et la manière de l'intégrer sous Zope, il convient d'observer l'aspect général de la page pour distinguer le contenant du contenu.

> **Remarque**
>
> *D'après ce que nous savons désormais des* `DTML Method` *et* `DTML Document`*, les éléments du contenant seront forcément des* `DTML Method` *(ou* `Python Script`*), et ceux du contenu ne pourront être que des* `DTML Document`*.*

Corps de la page

Dans notre page témoin, le plus important est le cadre « corps de la page ». Si nous devions créer une version imprimable de cette page, nous n'y inclurions probablement que cette partie, en la modifiant pour ne l'afficher qu'en noir et blanc. Ces éléments principaux seront des `DTML Document`, comme nous le verrons ultérieurement.

Feuille de style (CSS)

Le code HTML de la page témoin fait appel à une feuille de style (CSS). Nous allons donc créer un `DTML Document` comprenant cette CSS, que nous appelons `html_css`[1].

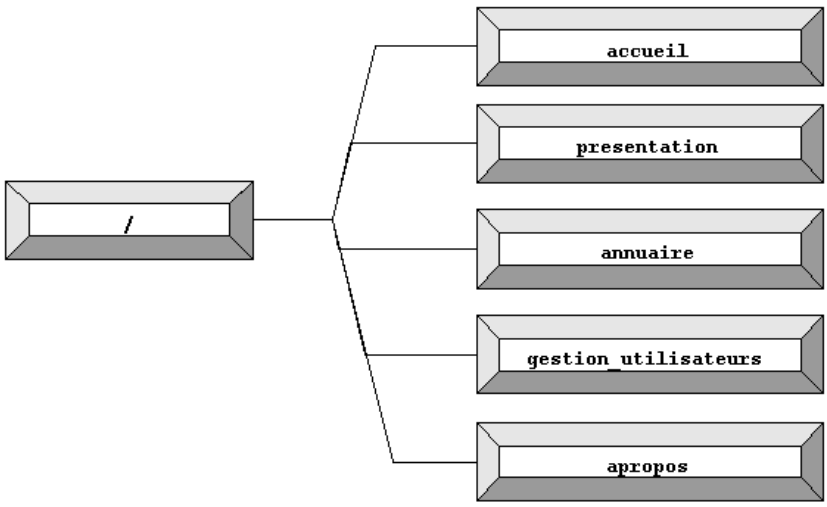
> **Remarque**
>
> *Pour la feuille de style CSS, nous avons créé un* `DTML Document` *et pas une* `DTML Method` *: considérer la feuille de style comme un « document » peut être contestable ; en revanche, utiliser les propriétés du document pour lui affecter certains paramètres, les couleurs, par exemple, s'avère fort pratique.*
>
> *Dans le cadre de notre étude, cela n'est qu'un point de détail : nous n'utiliserons pas de propriété avec notre feuille de style.*

1. Le code de cette feuille de styles est donné sur le site d'accompagnement du livre.

Barre de menus

La barre de menus, en haut de la page, reflète la structure du site. Il est probable que nous observions, dans l'arborescence de Zope, la structure suivante (voir figure 15-2) :

Figure 15-2
Arborescence générale du site



La barre de menus doit alors être générée automatiquement, en utilisant une propriété `titre-menu` des objets `Folder` correspondants pour en déterminer le titre.

Voici le code HTML correspondant à l'affichage du menu pour notre page témoin :

```
<TABLE width="100%" cellPadding="0" cellSpacing="0">
  <TR>
    <TD valign="bottom" class="barremenu">
      <TABLE cellpadding="0" cellspacing="0">
        <TR>
          <TD class="barremenu"> <A class=barremenu href=
          ➥"http://www.mouli.net/">Accueil</A>  </TD>
          <TD class="barremenu"><A class=barremenu href=
          ➥"http://www.mouli.net/">Présentation</A>  </TD>
          <TD class="barremenu"><A class=barremenu href=
          ➥"http://www.mouli.net/">Annuaire</A>  </TD>
          <TD class="barremenu"><A class=barremenu href=
          ➥"http://www.ailleurs.com/">A propos...</A></TD>
        </TR>
      </TABLE>
    </TD>
  </TR>
</TABLE>
```

Ce code peut être facilement transcrit en DTML. Dans la racine du site, créer une méthode `html_barremenu` contenant le code suivant :

```
<TABLE width="100%" cellPadding="0" cellSpacing="0">
<TR>
<TD valign="bottom" class="barremenu">
<TABLE cellPadding="0" cellSpacing="0">
<TR>
<dtml-in "_['weballage.com'].objectValues(['Folder'])"
 skip_unauthorized sort=id mapping>
  <dtml-if "'titremenu' in _['sequence-item'].propertyIds()">
    <TD class=barremenu><A class=barremenu href="<dtml-var absolute_url>">
    ➥<dtml-var titremenu></A>
        </TD>
  </dtml-if>
</dtml-in>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
```

Créer ensuite les dossiers correspondants : `01_accueil`, `02_presentation`, `04_annuaire` et `05_apropos` et donner un titre à chaque dossier dans la propriété `titremenu`. La numérotation elle-même a peu d'importance, l'essentiel est que les identifiants soient ordonnés. En se rendant à l'URL *http://localhost:8080/weballage.com/html_barremenu*, on peut déjà observer le résultat : la barre de menus s'affiche correctement.

Remarque

L'accès à l'URL http://localhost:8080/weballage.com/html_barremenu ne fait pas appel à la feuille de style `html_css`.

Préfixer les objets par un nombre permet de contrôler l'ordre dans lequel ils sont affichés. De plus, grâce à l'option `skip_unauthorized` de la balise`<dtml-in>`, la sécurité est d'ores et déjà gérée : un dossier pour lequel l'utilisateur n'a pas les droits suffisants n'apparaît pas. Enfin, un dossier qui n'a pas de propriété `menutitre` n'apparaît pas non plus : cela peut s'avérer pratique pour gérer des dossiers particuliers ; nous en verrons un exemple en matière de gestion de la sécurité ou des actualités.

Remarque

Nous avons utilisé la méthode `propertyIds` *associée à l'attribut* `mapping` *de la balise* `<dtml-in>`*, car le simple test* `<dtml-if titremenu>` *n'aurait pas fonctionné dans notre cas en raison de l'acquisition. L'utilisation de* `propertyIds` *permet d'inhiber l'acquisition lors de la recherche de l'attribut* `titremenu`.

Éléments répétitifs

Présentation des boîtes

De nombreux éléments répétitifs apparaissent sur notre page témoin. D'abord, les actualités : chaque élément d'actualité est doté de la même structure, et les quatre éléments les plus récents doivent être affichés.

De même, la page se décompose en deux parties qui peuvent être identifiées par leur titre (« suivi des commandes » et « informations complémentaires »). Là encore, ces éléments devront être mis en pages automatiquement pour notre site.

La génération des boîtes, notamment, est très répétitive. Cela vaut la peine de généraliser cet élément avec un `Python Script`. Voici par exemple le code HTML pour une boîte d'actualité :

```
<TR>
   <TD width="100%" class="actutitre">
      <TABLE width="100%" border=0 cellpadding="0"
      ➥cellspacing="0" class="actucontenu">
         <TR>
            <TD width="100%" colspan="3" class="actucontenu"
               ><A class="actucontenu" href="http://www.autrepart.xyz/">
               ➥<IMG alt="Info Extranet" border="0" align="left" src="img
               ➥_actuste.gif">Mise à jour du site web</A><BR>
               01/01/2000 16:34<BR>Le site web a été mis à jour aujourd'hui :
               ➥venez profiter de ses nombreuses améliorations ! </
            TD>
         </TR>
      </TABLE>
   </TD>
</TR>
```

Conseil

Le langage HTML est très (trop ?) sensible aux sauts de ligne, en particulier dans les tableaux : un saut de ligne juste après une balise `<TD>` ou juste avant une balise `</TD>` peut engendrer des effets de bord déplaisants selon le navigateur utilisé. Pour éviter cela, deux solutions : la première consiste à ne pas indenter le code pour ne pas ajouter de saut de ligne, la seconde consiste à placer le saut de ligne à l'intérieur de la balise, comme dans l'exemple précédent. Le code a alors l'apparence suivante :

```
<TD
  >Contenu de la cellule</
  TD>
```

Cette astuce a en outre le mérite de fonctionner sur tous les navigateurs !

Dans le code de cette boîte, seuls sont différents les classes CSS du fond et du bord de la boîte, le contenu lui-même et le *padding*, c'est-à-dire l'espace qui se trouve entre le bord de la boîte

et le contenu. Ces paramètres seront donc transmis en paramètres à un script `html_boite` situé dans la racine de `weballage.com`, et contenant le code suivant :

```
return """<TR>
  <TD width="100%%" class="%s">
    <TABLE width="100%%" border="0" cellpadding="%d" cellspacing="0" class="%s">
      <TR>
        <TD width="100%%" colspan="3" class="%s"
          >%s</
          TD>
      </TR>
    </TABLE>
  </TD>
</TR>""" % (classebords, padding, classecontenu, classecontenu, contenu)
```

Utiliser ensuite l'onglet `Test` pour vérifier le bon fonctionnement du script.

Attention

Ce script génère du code HTML : le résultat de la page de test sera donc probablement incohérent. Utilisez la fonction Afficher la source de votre navigateur pour vérifier que le code HTML généré correspond bien au résultat souhaité.

Parcours du contenu

Au-delà des observations que nous venons de faire à propos des boîtes, le contenu même de la page est répétitif : chaque actualité, chaque boîte de contenu, chaque boîte de la zone « information », peut contenir un ou plusieurs éléments. Chaque fois qu'un cas similaire se présente, la démarche doit être la même que celle adoptée pour la barre de menus : il faut placer dans un dossier les éléments à parcourir et utiliser une boucle `<dtml-in>` pour parcourir et afficher chacun de ces éléments.

Dans le cas de la barre de menus, chaque élément était lui-même un dossier doté de la propriété `titremenu`. Dans le cas de la zone centrale, les éléments à parcourir seront des pages saisies par les utilisateurs. Dans un premier temps, nous considérerons qu'il ne s'agit que de `DTML Document` puis, lorsque nous aurons créé les objets pour les autres types de contenu (entrées d'annuaire, actualités…), nous ajouterons le méta-type de ces objets à notre boucle de parcours.

La zone du milieu de la page n'est pas la seule concernée par ce processus : il faut donc disposer d'un moyen qui permette de distinguer, pour chaque objet, l'emplacement où il doit être rangé. Pour chaque page consultable, nous allons créer un dossier `contenu` pour conserver ces objets.

Commencer par créer un dossier `arch_contenu` dans le dossier `weballage.com` : logiquement, il contiendra le contenu de la zone centrale pour la page d'accueil. Puis, créer un `DTML Document` appelé `01_bienvenue` (nous utilisons là encore un préfixe numérique pour trier les entrées), contenant un simple message de bienvenue. Donner la valeur « `Bienvenue` » à la propriété `title`. Créer un autre `DTML Document` appelé `02_quisommesnous`, ayant la valeur « `Qui sommes-nous ?` » pour `title`, et contenant du texte à votre convenance.

Puis, dans le dossier `weballage.com`, créer une méthode `html_contenu`, chargée de parcourir les éléments du dossier `arch_contenu` et de les afficher un à un dans une boîte :

```
<!-- CONTENU DE LA PAGE -->
<TD valign="top" width="100%">
  <TABLE border="0" cellspacing="2" cellpadding="1" width="100%">

  <dtml-if arch_contenu>
    <dtml-in "arch_contenu.objectValues(['DTML Document'])" sort="id"
   skip_unauthorized>
       <dtml-if title>
           <dtml-var "html_boite('contenu', 'contenutitre', title, 1)">
       </dtml-if>
       <dtml-var "html_boite('contenutitre', 'contenu', _['sequence-item'], 5)">
       <TR><TD> </TD></TR>
    </dtml-in>
  </dtml-if>

  </TABLE>
</TD>
```

Là encore, pour se rendre vraiment compte du résultat, afficher le source de la page HTML résultante.

Si un objet parcouru a une propriété `title` définie, une barre de titre sera générée avant le contenu proprement dit.

On notera au passage que si le dossier courant ne contient pas de dossier `arch_contenu`, l'acquisition fera que le dossier `arch_contenu` d'un parent sera rendu. Si aucun parent ne comprend un tel dossier, un tableau vide sera renvoyé.

Barre de gauche

La partie gauche de la page peut être décomposée en trois zones : actualités, aide et informations, chacune étant elle-même découpée en d'autres éléments. Nous allons encore une fois créer une `DTML Method` pour gérer cette partie du site, que nous appellerons `html_barre_gauche`. Ici, cependant, nous ne pouvons pas afficher directement des boîtes, car chaque élément de la barre de gauche a une structure fondamentalement différente : il nous faut les traiter au cas par cas.

La méthode contient le code suivant :

```
<TR>
  <!-- BARRE DE GAUCHE -->
    <TD vAlign="top" width="100">
      <TABLE border=0 cellpadding="1" cellspacing="2">
        <dtml-var html_actubarre>
        <TR><TD> </TD></TR>
        <dtml-var html_aidebarre>
```

```
        <TR><TD> </TD></TR>
        <dtml-var html_infobarre>
      </TABLE>
    </TD>
</TR>
```

Nous étudierons en détail les méthodes `html_actubarre`, `html_aidebarre` et `html_infobarre` plus avant dans ce chapitre. Pour l'instant, ces trois `DTML Method` contiennent uniquement ce qui permet d'afficher des boîtes vides. Ainsi, pour `html_actubarre` :

```
<!-- ACTUALITES -->
<dtml-var "html_boite('actucontenu', 'actutitre', 'Actualités', 1)">
<dtml-var "html_boite('actutitre', 'actucontenu', 'Ceci contiendra une nouvelle
➥fraîche.', 1)">
```

Pour `html_aide`, en revanche, nous allons créer des boîtes à partir des `DTML Document` situés dans le dossier `arch_aide` (qu'il faut créer) :

```
<!-- AIDE -->
<dtml-if arch_aide>
  <dtml-in "arch_aide.objectValues(['DTML Document'])" skip_unauthorized>
    <dtml-if sequence-start>
      <dtml-var "html_boite('aidecontenu', 'aidetitre', 'Aide', 1)">
    </dtml-if>
   <dtml-var "html_boite('aidetitre', 'aidecontenu', _['sequence-item'], 1)">
  </dtml-in>
</dtml-if>
```

La gestion des informations diverses présente une petite particularité : il est préférable de ne pas lister que les informations du dossier courant, mais toutes les informations depuis la racine du site, pour, au fur et à mesure du parcours, afficher des informations supplémentaires, tout en conservant des informations primordiales (par exemple, le nom de l'utilisateur connecté devrait apparaître sur toutes les pages : il doit donc être géré à la racine du site).

Voici le code correspondant :

```
<!-- INFOS -->
<dtml-if arch_info>
  <dtml-var "html_boite('infocontenu', 'infotitre', 'Info', 1)">
  <dtml-in "PARENTS" reverse skip_unauthorized>
    <dtml-if "'arch_info' in objectIds()">
      <dtml-in "arch_info.objectValues(['DTML Document'])" skip_unauthorized>
     <dtml-var "html_boite('infotitre', 'infocontenu', _['sequence-item'], 1)">
      </dtml-in>
    </dtml-if>
  </dtml-in>
</dtml-if>
```

Là encore, créer un dossier arch_info que l'on peuplera de quelques DTML Document. Ici, les titres des documents ne sont pas pris en compte.

Titre de la page

Le titre de la page apparaît dans la barre de titre de la fenêtre du navigateur. Il serait à la rigueur possible de la faire porter par une propriété title du dossier correspondant à la page courante, mais cela nous priverait de la possibilité d'utiliser du code DTML pour calculer ce titre.

Créer plutôt un document html_titre, contenant simplement le texte « weballage.com : accueil ». Nous nous servirons de ce document avec la balise <TITLE> du code HTML final.

Cartouche

Il manque un détail dans notre représentation des documents de la partie centrale : le nom de l'auteur et la date de la dernière modification. L'auteur est en réalité celui qui possède l'objet (rôle Owner). Pour éviter de surcharger le document, et surtout afin de pouvoir utiliser ce procédé en d'autres endroits du site, créer un document html_cartouche, dans la racine de weballage, contenant le code suivant :

```
<BR>
<p class="cartouche" align="right">
  <dtml-with owner_info mapping>
    Créé par <a href="mailto:<dtml-var id>@weballage.com"><dtml-var
             id>@weballage.com</a>.
  </dtml-with>
  Dernière modification le <dtml-var "bobobase_modification_time()
  ➥.strftime('%d/%m/%Y %H:%M:%S')">
</p>
```

La date est ici formatée en français, et l'adresse e-mail de l'auteur est affichée (il s'agit en réalité de son nom d'utilisateur concaténé à « @weballage.com »).

Le problème de ce code tient à ce que l'attribut owner_info n'est accessible qu'aux administrateurs. Pour y avoir accès indépendamment du type d'utilisateur (anonyme ou pas), cliquer sur l'onglet Proxy de html_cartouche et donner à la méthode le rôle Proxy de Manager.

Pour intégrer le cartouche dans les documents courants, modifier le document html_contenu et remplacer la ligne :

```
    <dtml-var "html_boite('contenutitre', 'contenu', _['sequence-item'], 5)">
```

par :

```
    <dtml-var "html_boite('contenutitre', 'contenu', _['sequence-item']
    ➥+ _['html_cartouche'], 5)">
```

Images et logo

La barre de titre comporte deux images : on peut les télécharger depuis *www.weballage.com*, ou, à défaut, utiliser des images de petite taille (largeur de 100 pixels pour celle de gauche, hauteur de 100 pixels pour celle du haut).

Au cas où une partie du site serait placée sous Apache, pour des questions de performances, créer un dossier `static` dans `/` (et pas dans `weballage.com`), et y placer les deux images, sous les noms `img_logo` et `img_bandeau`.

> **Remarque**
>
> *Le dossier* `static` *doit être créé à la racine de Zope (et donc être partagé par tous les sites virtuels de Zope) car, en cas de déplacement vers Apache, les* `AccessRules` *d'Apache permettront de spécifier un répertoire* `static` *différent pour chaque serveur virtuel (l'interaction entre Zope et Apache est commentée au chapitre 13 de notre ouvrage).*

Il reste enfin la partie supérieure de la page : le bandeau du haut, avec le logo. Suivant la partie du site dans laquelle se trouve l'utilisateur, il peut être nécessaire de changer de logo (par exemple, pour afficher un panier d'achats dans la partie « extranet »). Nous allons donc rédiger la méthode `html_barre_haut` comme suit :

```
<TD vAlign="top" width="100">
  <TABLE border=0 width="100" cellspacing="0" cellpadding="0">
    <TR>
      <TD><dtml-var html_logo></TD>
    </TR>
  </TABLE>
</TD>

<TD vAlign="top">
  <TABLE width="100%" height="130" cellspacing="0" cellpadding="0">
    <TR>
      <TD  width="100%" align="center"><dtml-var html_bandeau></TD>
    </TR>
    <TR>
      <TD valign="bottom"><dtml-var html_barremenu></TD>
    </TR>
  </TABLE>
</TD>
```

Il faut ensuite créer deux méthodes, `html_logo` et `html_bandeau`, contenant respectivement les textes suivants :

```
<IMG src="/static/img_logo.gif">
```

et

```
<IMG src="/static/img_bandeau.gif">
```

Intégration du code HTML

Nous avons conçu individuellement les parties de code HTML dont nous avons besoin en découpant la page témoin en différentes parties et en automatisant la procédure chaque fois qu'il nous était possible de le faire. Il ne reste donc plus qu'à intégrer ces différents éléments pour que le site produise sa première page HTML complète.

Pour des raisons simples à comprendre, la `DTML Method`, qui réunit tout le code HTML, n'est pas `index_html`. Par exemple, si nous souhaitons créer une interface pour imprimer des pages, comme nous l'avons mentionné plus haut, la méthode `index_html` devra changer de comportement pour afficher la page suivant une autre disposition. Pour éviter de mélanger contenu HTML et logique de traitement (dans ce cas, le code qui détermine le moment opportun d'afficher une page imprimable), il convient de créer une méthode intermédiaire, `html_rendu`, qui contient le code suivant :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE><dtml-var html_titre></TITLE>
    <META content="text/html; charset=windows-1252" http-equiv=Content-Type>
    <dtml-var html_css>
  </HEAD>
  <BODY bgColor="#ffffff">
    <TABLE border=0 width="100%" cellspacing="0" cellpadding="0">
      <TR>
        <dtml-var html_barre_haut>
      </TR>
      <TR>
        <dtml-var html_barre_gauche>
        <dtml-var html_contenu>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

On peut en observer le résultat sur *http://localhost:8080/weballage.com/html_rendu* : le site a pris forme physiquement.

Il ne reste plus qu'à établir l'architecture logique, et à créer la méthode `index_html` qui permettra de rendre automatiquement la page lors de l'accès à un dossier. Pour l'instant, pour tester le comportement du site, créer une méthode `index_html` contenant juste le code `<dtml-var html_rendu>`.

En voici le résultat (voir figure 15-3) :

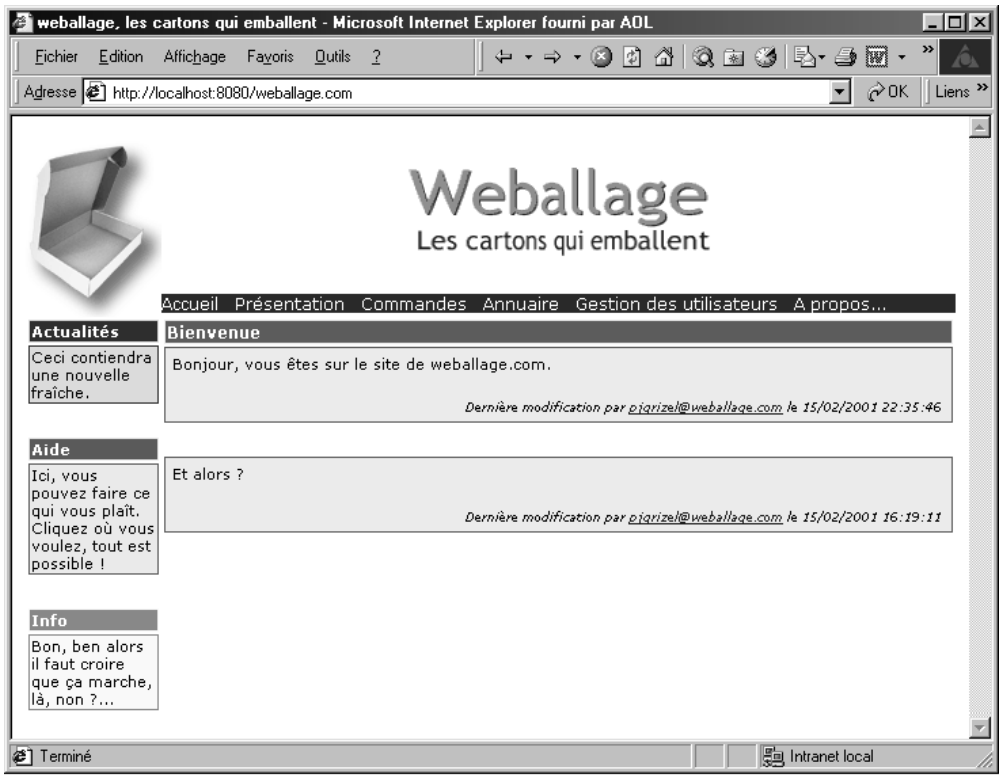


Figure 15-3
La page témoin intégrée dans Zope

Page imprimable

Toutefois, avant de passer à l'architecture logique, il convient de se pencher sur l'écriture du code qui permet de rendre une page imprimable. Pour cela, il nous faut utiliser une autre feuille de style et une autre méthode du style `html_rendu`. L'acquisition est ici idéale pour surcharger les deux documents en question. Créer un `dossier` appelé `imprimer` et y attacher une `DTML Method html_rendu` contenant le code suivant :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE><dtml-var html_titre></TITLE>
    <META content="text/html; charset=windows-1252" http-equiv=Content-Type>
    <dtml-var html_css>
```

```
  </HEAD>
  <BODY bgColor="#ffffff">
    <TABLE border=0 width="100%" cellspacing="0" cellpadding="0">
      <TR>
        <dtml-var html_contenu>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

Il s'agit en fait d'une version simplifiée de la méthode `html_rendu` de la racine du site. De même, surcharger `html_css` comme suit :

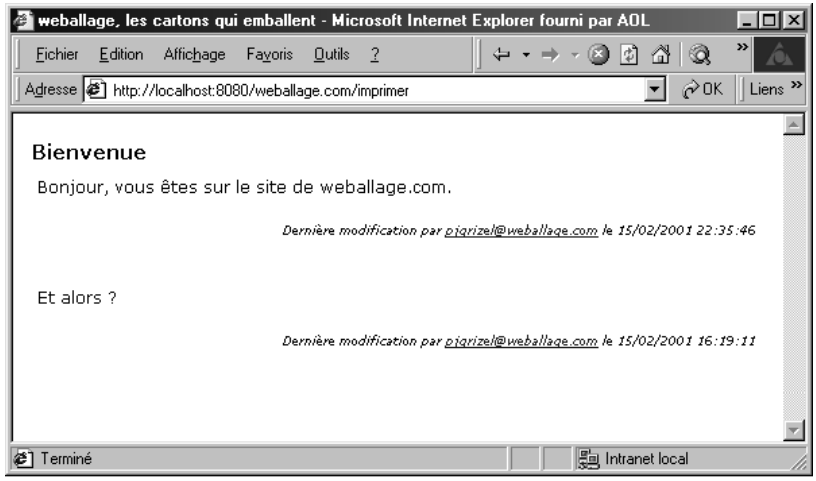
```
<STYLE type=text/css> <--
.contenutitre {
   BACKGROUND-COLOR: #ffffff; COLOR: #000000; FONT: bold 15px Verdana, Arial,
   ➥Helvetica, sans-serif; VERTICAL-ALIGN: top
}
.contenu {
   BACKGROUND-COLOR: #ffffff; COLOR: #000000; FONT: 12px Verdana, Arial, Helvetica,
   ➥sans-serif; VERTICAL-ALIGN: top
}
.cartouche {
   BACKGROUND-COLOR: #ffffff; COLOR: #000000; FONT: 9px Verdana, Arial,
 Helvetica, sans-serif; VERTICAL-ALIGN: top; FONT-STYLE: italic;
}
.hztable {
   COLOR: #ffffff; FONT: 9px Verdana, Arial, Helvetica, sans-serif
}
A.hztable:visited {
   COLOR: #ffffff; FONT-WEIGHT: bold; TEXT-DECORATION: none
}
A.hztable:link {
   COLOR: #ffffff; FONT-WEIGHT: bold; TEXT-DECORATION: none
}
TD.hztable {
   COLOR: #000000; FONT: 9px Verdana, Arial, Helvetica, sans-serif
}
TH.hztable {
   BACKGROUND-COLOR: #6699cc; COLOR: #ffffff; FONT: 9px Verdana, Arial,
 Helvetica, sans-serif; TEXT-ALIGN: center
}
.hztable_odd {
   COLOR: #000000; FONT: 9px Verdana, Arial, Helvetica, sans-serif;
   ➥VERTICAL-ALIGN: top
}
TR.hztable_odd {
```

```
   BACKGROUND-COLOR: #ffffee; FONT: 9px Verdana, Arial, Helvetica, sans-serif
}
.hztable_even {
   COLOR: #000000; FONT: 9px Verdana, Arial, Helvetica, sans-serif;
   ➥VERTICAL-ALIGN: top
}
TR.hztable_even {
   BACKGROUND-COLOR: #ffffff; FONT: 9px Verdana, Arial, Helvetica, sans-serif
}
-->
</STYLE>
```

Ici encore, il s'agit d'une version simplifiée du document `html_css` de la racine.

Désormais, on peut rajouter `/imprimer` à la fin de quelque URL que ce soit du site, et une version imprimable de la page s'affiche alors, sans éléments graphiques (voir figure 15-4).

Figure 15-4
Page imprimable



Bilan

Voici, pour résumer, une vue de l'architecture de présentation du site web que nous avons créé (voir figure 15-5).

L'architecture peut paraître compliquée mais, en fait, elle est très simple à utiliser : il suffit de créer des `Folder` et des `DTML Document` contenant le strict nécessaire – le contenu – et de surcharger ponctuellement certains éléments si besoin est, comme nous l'avons fait pour la page imprimable. De même, il est possible de surcharger `html_css` pour avoir une autre feuille de style dans une partie du site ; de surcharger `html_logo` et `html_bandeau` pour utiliser d'autres

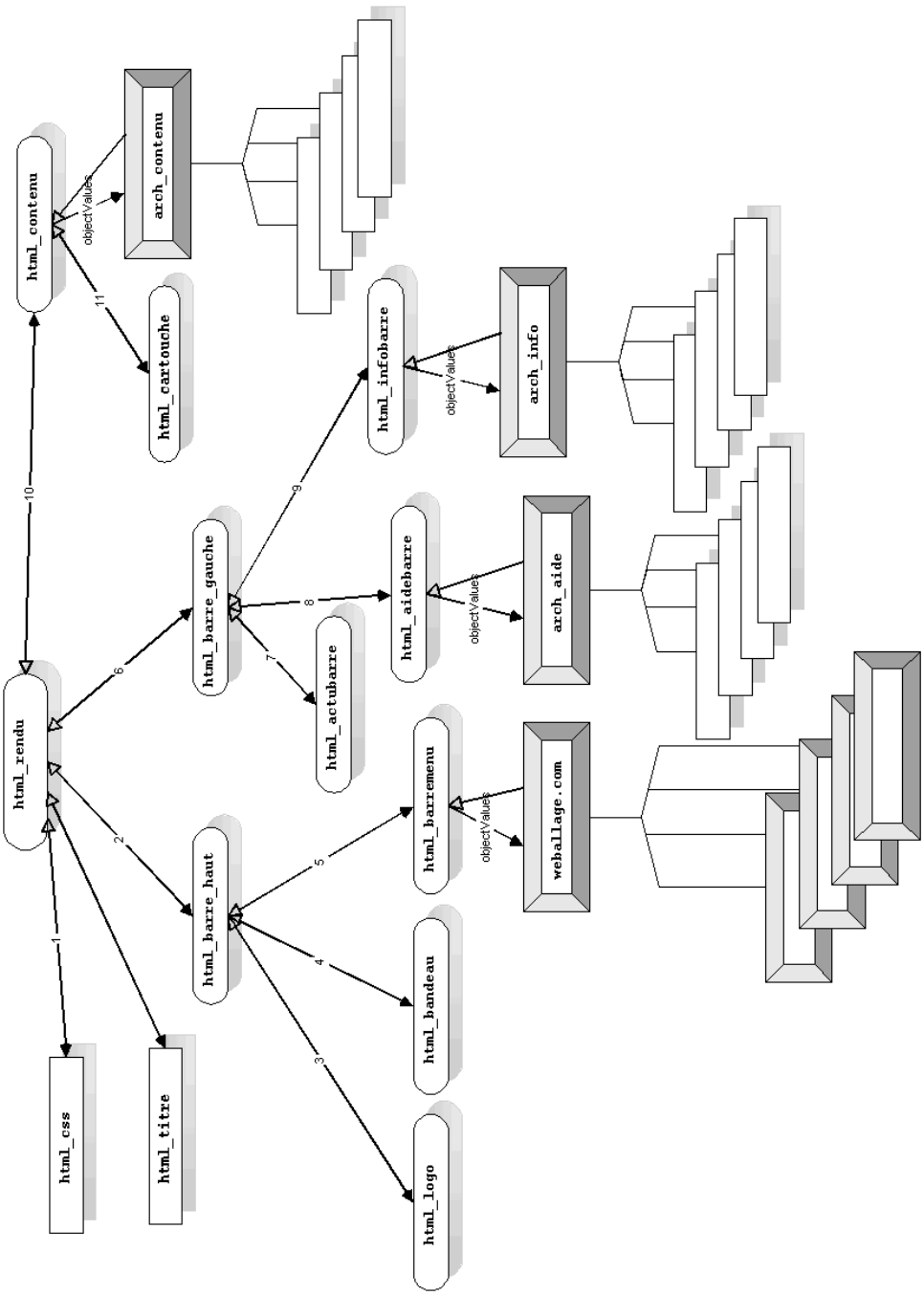


Figure 15-5
Architecture de présentation

images, au sein de l'intranet, par exemple. Mais aussi, de surcharger `html_infobarre` pour modifier l'affichage de certaines informations.

De même, il se trouve parfois une âme contestataire pour arguer de ce que la barre contenant les boîtes « actualités », « aide » et « informations » aurait été plus jolie à droite qu'à gauche. Là encore, il suffit de remplacer, dans `html_rendu`, les lignes :

```
<TR>
  <dtml-var html_barre_gauche>
  <dtml-var html_contenu>
</TR>
```

par :

```
<TR>
  <TD colspan="2">
    <TABLE border="0" cellspacing="0" cellpadding="0" width="100%">
      <TR>
        <dtml-var html_contenu>
        <dtml-var html_barre_gauche>
      </TR>
    </TABLE>
  </TD>
</TR>
```

Le résultat est le suivant (voir figure 15-6). En effet, les boîtes sont nettement mieux à droite.

La modification, qui peut être effectuée à des milliers de kilomètres grâce à l'EDI de Zope, a pris tout au plus 2 minutes. Le résultat ne convient pas ? Un clic sur Undo et on revient à l'état précédent.

Dans l'architecture que nous avons adoptée, nous aurions pu aller plus loin dans la séparation contenant/contenu ; néanmoins, ç'eût été au détriment d'une relative clarté du schéma. Nous obtenons ici un découpage facile à comprendre (chaque zone de la page est rendue par une `DTML Method` différente), et développer plus encore le schéma offrirait un niveau de détail inutile dans le cadre de notre étude de cas. Des projets de grande envergure, où le comportement du site change radicalement au fur et à mesure de son exploration, peuvent cependant requérir plus de détail de mise en œuvre.

D'autres améliorations peuvent être apportées au modèle, mais ce serait une gageure que de prétendre faire le tour des possibilités de Zope : le lecteur aura soin de l'explorer selon son goût, ses idées, ses besoins.

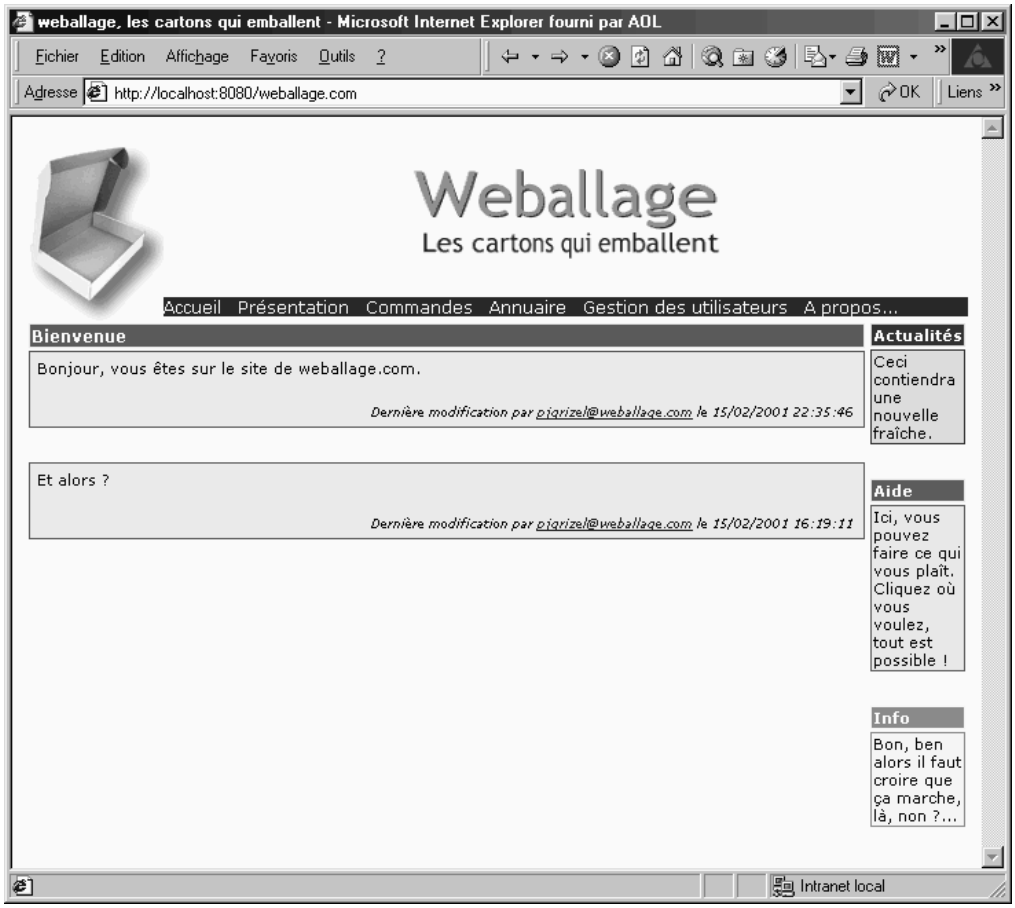


Figure 15-6
Les boîtes à droite

Architecture logique

L'architecture logique correspond strictement au modèle présenté au chapitre 14 (voir page 437).

Gestion des utilisateurs

Rôles

Il convient, avant de commencer à réaliser le site proprement dit, de créer les différents rôles dont nous allons avoir besoin :

- administration ;
- intranet ;
- extranet.

Créez ces trois rôles depuis l'onglet Security du dossier `weballage.com`.

Utilisateurs

Pour effectuer les tests de sécurité, il nous faut quelques utilisateurs de test : créer un `User Folder` dans `weballage.com` et ajouter les utilisateurs *`testintranet`*, *`testextranet`* et *`test-administration`*, chacun doté du rôle correspondant (respectivement, intranet, extranet, et administration). Dans la pratique, les utilisateurs dotés du rôle administration auront également le rôle intranet (puisqu'ils font partie de l'entreprise) : on donnera donc également le rôle intranet à l'utilisateur *`testadministration`*.

Connexion

Dans notre cas, la gestion des utilisateurs va simplement consister à présenter à un utilisateur non connecté une boîte lui permettant de rentrer dans la partie privée du site (dans la barre d'informations, à gauche).

L'affichage de la boîte de gauche est effectué depuis le dossier `arch_info` situé sous `weballage.com`. Créer un dossier `01_connexion` contenant le code suivant :

```
<dtml-if "AUTHENTICATED_USER.getRoles() == ('Anonymous', )">
  <FORM action="" method="POST">
    Vous n'êtes pas identifié. Vous pouvez rentrer dans l'Intranet ou
    l'Extranet en cliquant sur le bouton ci-dessous.
    <center><input type="submit" name="proc_connecter:method"
             value="Entrer"></center>
  </FORM>
<dtml-else>
  Bonjour, <dtml-var AUTHENTICATED_USER>.
</dtml-if>>
```

Si l'utilisateur ne possède que le rôle anonyme, on lui propose un bouton pour se connecter. S'il est déjà connecté, on affiche un message de bienvenue.

Le formulaire de connexion renvoie vers un dossier `proc_connecter`. Créer ce dossier dans `weballage.com` et changer simplement ses options de sécurité : décocher la case « Acquire permission settings » pour les permissions `Access content information` et `View`. De même, pour ces deux permissions, cocher les rôles administrateur, intranet et extranet.

C'est tout : le code fonctionne correctement sans que l'on doive écrire la moindre ligne. Ouvrez une nouvelle instance de votre navigateur et allez sur la page d'accueil : le bouton Entrer apparaît. Cliquez : la boîte classique demandant le nom d'utilisateur et le mot de passe apparaît. Une fois que vous êtes connecté, la page d'accueil est affichée, et votre nom d'utilisateur apparaît dans la boîte d'informations, en bas à gauche.

Il est possible de présenter un message de bienvenue à un utilisateur qui se connecte : pour cela, on crée un dossier `arch_contenu` dans `proc_connecter` et on y place les `DTML Document` correspondant aux messages que l'on souhaite afficher.

Actualités

La première partie « fonctionnelle » du site que nous allons implémenter concerne la gestion des actualités. Nous allons d'abord créer la ZClasse correspondante et l'associer à notre architecture documentaire. Puis, nous écrirons l'interface de création de nouvelles actualités.

Création de la ZClasse

Nous allons créer une ZClasse, `weballage_actu`, selon la procédure classique (voir figure 15-7) : on se place dans la section Products du panneau de configuration de Zope, puis on clique sur Add product. On crée un produit appelé `weballage_actu`, puis, au sein de ce produit, une ZClasse appelée elle aussi `weballage_actu` et dotée des caractéristiques suivantes :

`Id`	`weballage_actu`
`Title`	`Actualité Weballage`
`Meta Type`	`Actualite Weballage`
`Base classes`	`OFS:Folder`
`Create Constructor Objects?`	`Oui`
`Include standard Zope persistent object base classes`	`Oui`

Pourquoi dériver notre classe de `Folder` ? Pour simplifier son intégration à l'architecture documentaire. Si chaque élément d'actualité est un `Folder`, il est possible de surcharger l'un des éléments de notre architecture pour *une* nouvelle en particulier. Si nous ne dérivions pas notre ZClasse de `Folder`, il serait difficile d'y parvenir.

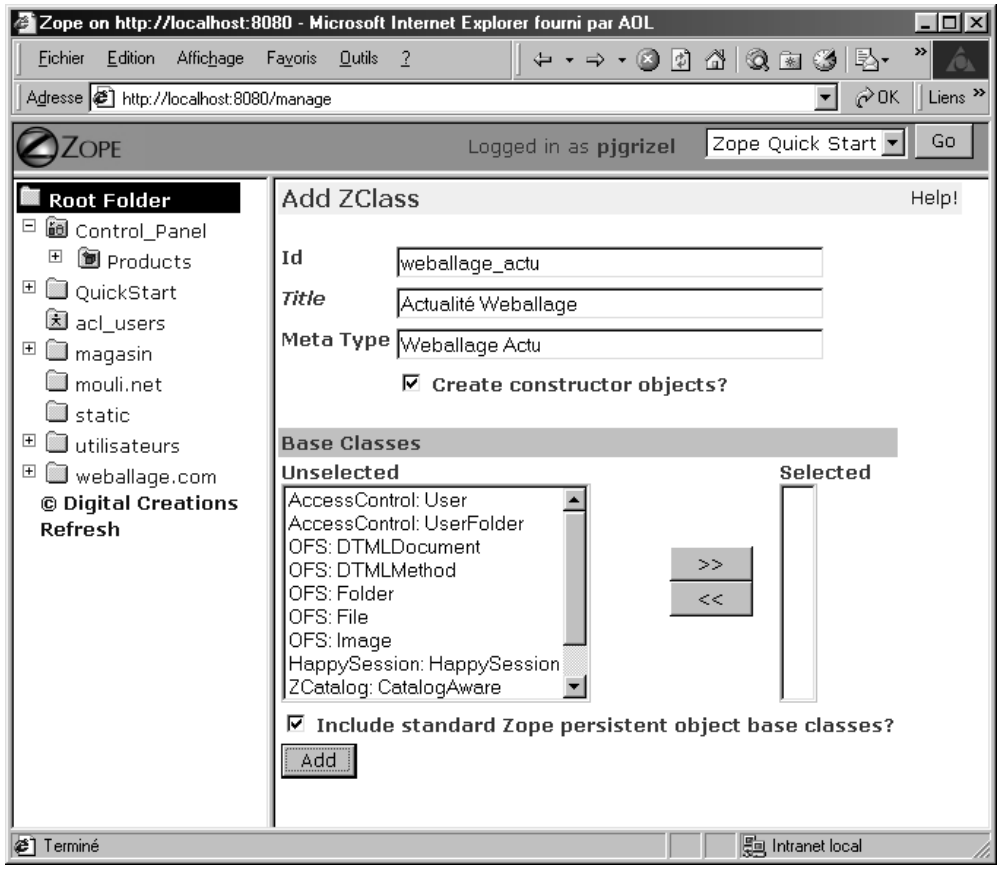


Figure 15-7
Création de la ZClasse, `weballage_actu`

Créer ensuite une `PropertySheet` appelée `proprietes` et lui affecter les propriétés suivantes :

Nom	Type	Valeur
Titre	*String*	*(aucune)*
Date	*Date*	*01/01/2001 (une valeur par défaut est obligatoire pour les champs de type Date)*
Resume	*String*	*(aucune)*
Contenu	*Text*	*(aucune)*

Il faut ensuite associer ces propriétés à une vue : nous allons remplacer la vue `Properties` du `Folder`. Pour ce faire, on clique sur l'onglet View de la ZClasse et on associe la vue `Property` à `propertysheets/proprietes/manage`.

Affichage du résumé

Créer dans la ZClasse la `DTML Method` permettant d'afficher le résumé de l'élément d'actualité en HTML, appelée `html_resume` :

```
<dtml-if css_class>
  <dtml-call "REQUEST.set('weballage_actu_css_class', 'class=\x22%s\x22'
  ➥% css_class)">
<dtml-else>
  <dtml-call "REQUEST.set('weballage_actu_css_class', '')">
</dtml-if>

<A <dtml-var weballage_actu_css_class> href="<dtml-var "absolute_url()">">
<dtml-var Titre></A><BR>
<dtml-var "Date.strftime('%d/%m/%Y')"><BR>
     <dtml-var Resume>
```

Remarque

Nous avons utilisé des feuilles de style dans notre page HTML ; la classe correspondant à la boîte d'actualités est en principe `actucontenu`*, mais nous ne devons pas inclure directement cette information dans la ZClasse : le code du début permet d'utiliser la variable* `css_class` *pour la classe de notre élément, si cette variable est définie ; dans le cas contraire, aucune classe CSS n'est mentionnée dans le code HTML.*

De plus, l'utilisation de guillemets étant nécessaire pour encadrer la valeur de l'attribut `class` *de la balise générée, nous utilisons* `\x22` *(22 hexa étant le code ASCII du guillemet) pour les insérer dans la variable* `weballage_actu_css_class`*.*

Affichage du contenu

Créer ensuite une `DTML Method html_actu_content` dans la ZClasse, contenant le code qui permet d'afficher un élément d'actualité dans la zone « contenu » :

```
<dtml-if css_class>
  <dtml-call "REQUEST.set('weballage_actu_css_class', 'class=\x22%s\x22' %
   css_class)">
<dtml-else>
  <dtml-call "REQUEST.set('weballage_actu_css_class', '')">
</dtml-if>
<P <dtml-var weballage_actu_css_class> align="center"><B><dtml-var Resume></B></P>
<P <dtml-var weballage_actu_css_class>><dtml-var Contenu fmt="structured-text"></P>
```

Intégration du résumé dans le site

Pour étudier le comportement de la ZClasse, créer, sous le dossier `weballage.com`, un dossier `arch_actualites`, dans lequel on peut placer quelques objets `Weballage Actu` ; éditer leurs propriétés pour vérifier le bon fonctionnement de l'interface d'administration de la classe et de la méthode `html_resume`. Par exemple, si on crée un objet appelé `actu_01`, on peut vérifier que la méthode `html_resume` fonctionne correctement en accédant à l'URL *http://localhost:8080/weballage.com/arch_actualites/actu_01/html_resume*. Pour tester le bon comportement de la gestion de la feuille de style, essayer *http://localhost:8080/weballage.com/arch_actualites/actu_01/html_resume?css_class=actucontenu* et s'assurer dans le code source de la page résultante de la présence de `class="contenu"`.

Il convient ensuite de modifier la méthode `html_actubarre` de la racine du site pour qu'elle affiche les trois éléments les plus récents. Remplacer son code par le code suivant :

```
<!-- ACTUALITES -->
<dtml-var "html_boite('actucontenu', 'actutitre', 'Actualités', 1)">
<dtml-let css_class="'actucontenu'">
  <dtml-in "arch_actualites.objectValues('Weballage Actu')" skip_unauthorized
 reverse orphan="0" size="3" sort="Date">
    <dtml-var "html_boite('actutitre', 'actucontenu', _['html_resume'], 1)">
  </dtml-in>
</dtml-let>
```

Remarque

La présence de l'attribut `orphan` *dans la balise* `<dtml-in>` *permet de faire en sorte que la boucle soit parcourue trois fois au plus. Si* `orphan` *n'est pas précisé, sa valeur par défaut est 1 et quatre éléments pourront être affichés dans le pire des cas (si le dossier* `arch_actualites` *contient 4 objets* `Weballage Actualite`*).*

Et voici le résultat, sur la page d'accueil (voir figure 15-8) :

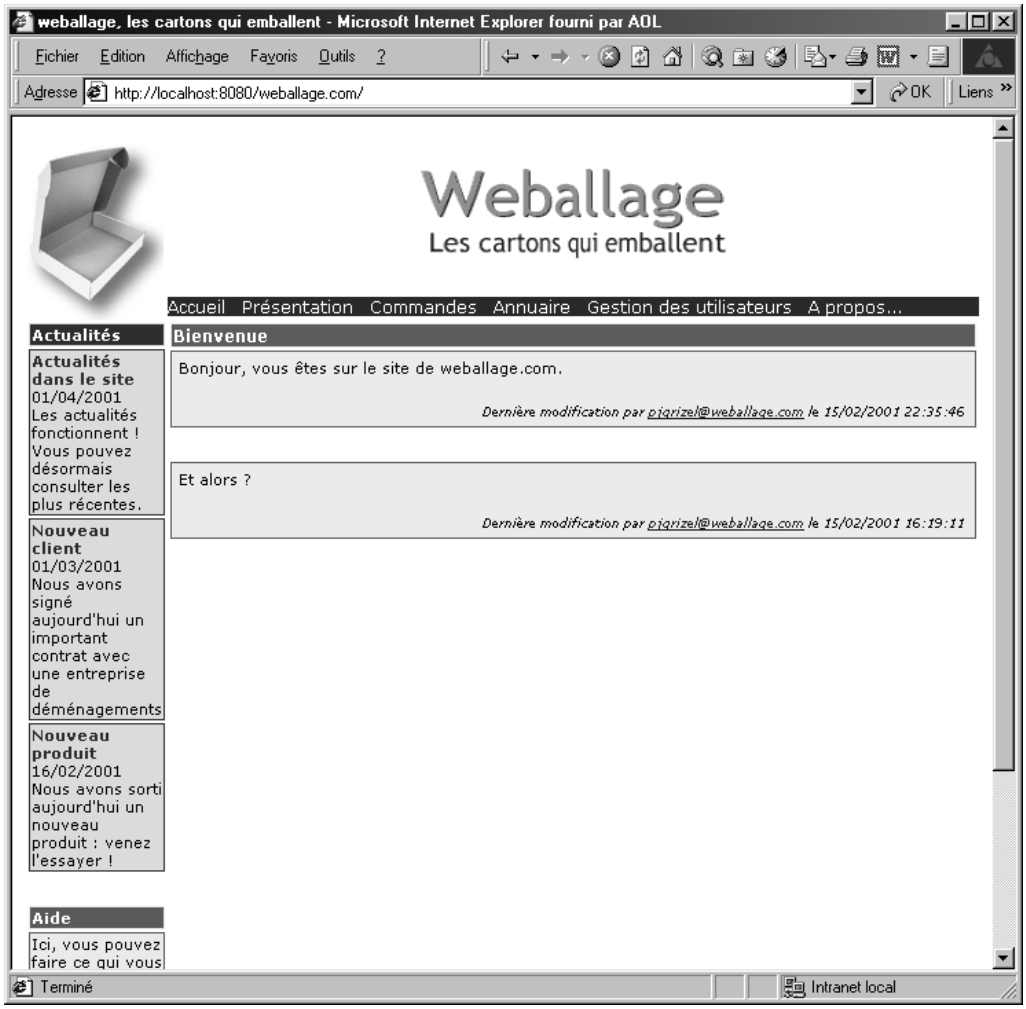


Figure 15-8
Affichage de la barre d'actualités

Intégration du contenu dans le site

Si, dans la page que nous venons de présenter, on clique sur le titre d'une des actualités, on obtient une page strictement identique, bien que le navigateur affiche l'URL de l'élément d'actualité : il nous faut surcharger `html_content` pour afficher le contenu réel de l'élément sélectionné, et créer une méthode dans notre ZClasse pour afficher son contenu préformaté.

Puis, on copie le document `html_content` de la racine du site dans le dossier `arch_actualites` et on remplace son contenu par le suivant :

```
<!-- CONTENU D'UNE ACTUALITE -->
<dtml-if html_actu_contenu>
  <TD valign="top" width="100%">
    <TABLE border="0" cellspacing="2" cellpadding="1" width="100%">
      <dtml-var "html_boite('contenu', 'contenutitre', Date.strftime
   ('%d/%m/%Y') + ' - ' + Titre, 1)">
      <dtml-let css_class="'contenu'">
        <dtml-var "html_boite('contenutitre', 'contenu', _['html_actu_contenu']
        + _['html_cartouche'], 5)">
      </dtml-let>
    </TABLE>
  </TD>
</dtml-if>
```

Ce code fait appel à la méthode `html_actu_contenu` que nous avons créée dans notre ZClasse. On peut désormais cliquer sur le titre d'une nouvelle dans la barre de gauche pour obtenir la page suivante (voir figure 15-9) :

Figure 15-9
Affichage d'une nouvelle complète

Gestion de la liste complète des actualités

Il est bien pratique de disposer d'une liste exhaustive des actualités présentes sur le site, sous forme de tableau, par exemple. Nous allons donc créer les méthodes et documents nécessaires.

Il faut tout d'abord que cette page soit accessible depuis la barre de menus, juste après l'entrée « Présentation » : créer un dossier appelé `02b_Actualites` et doté d'une propriété `titremenu`

contenant la chaîne « Actualités ». L'entrée s'ajoute automatiquement dans le menu, à l'emplacement souhaité (grâce au tri sur l'`id` des dossiers).

Dans ce dossier, nous pouvons utiliser l'architecture classique découpée en `DTML Document` dans le dossier `arch_contenu` : créer le dossier `arch_contenu` et y placer un `DTML Document` appelé `01_liste_actu`, intitulé « Liste des actualités » et contenant le code suivant :

```
<table class="hztable" border="0" cellspacing="1" cellpadding="1">
  <tr class="hztable">
    <th class="hztable">Date</th>
    <th class="hztable">Titre</th>
  </tr>

  <dtml-in "arch_actualites.objectValues(['Weballage Actu'])" reverse sort=
  ➥"Date">
    <dtml-if sequence-odd>
      <dtml-call "REQUEST.set('table_class_', ' class=\x22hztable_odd\x22 ')">
    <dtml-else>
      <dtml-call "REQUEST.set('table_class_', ' class=\x22hztable_even\x22 ')">
    </dtml-if>
    <tr <dtml-var table_class_>>
      <td <dtml-var table_class_>><dtml-var "Date.strftime('%d/%m/%Y')"></td>
      <td <dtml-var table_class_>><A HREF="<dtml-var absolute_url>"><dtml-var
      Titre></A></td>
    </tr>
  </dtml-in>
</TABLE>
```

Enfin, pour parfaire notre page, il convient d'ajuster le texte d'aide : créer, dans le dossier `02b_actualites`, un dossier `arch_aide`. Dans ce dossier, créer un `DTML Document` appelé `01` et contenant le texte suivant :

```
Cliquez sur un titre pour voir l'actualité correspondante.
```

Le résultat est présenté dans la figure ci-après (voir figure 15-10). Il suffit de cliquer sur un titre pour consulter l'actualité correspondante.

De même, on peut créer, dans le dossier `arch_actualites`, la boîte d'aide correspondante, et une boîte d'informations (dans un dossier `arch_info`) contenant un lien vers la page qui présente la liste des actualités. On peut par exemple utiliser le code suivant dans un `DTML Document` appelé `01` :

```
Vous pouvez consulter la <A class="infocontenu" href="<dtml-var
➥"_['02b_actualites'].absolute_url()">">liste complète des actualités</A>
```

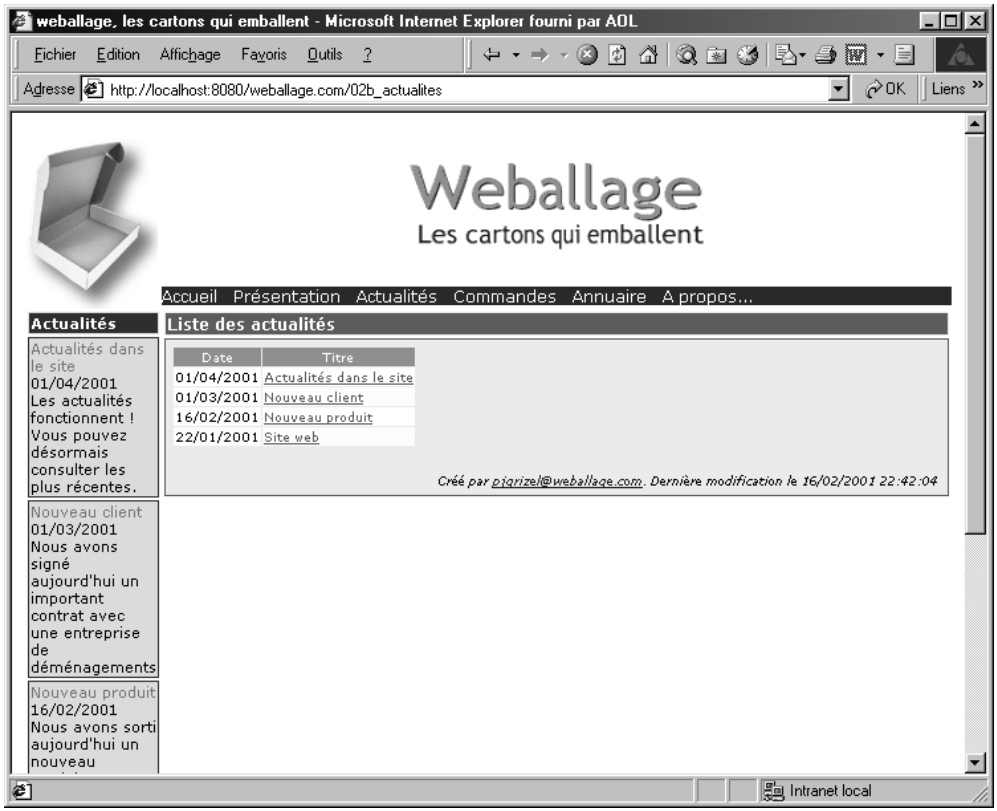


Figure 15-10
Affichage de la liste des actualités

Création et suppression

Il reste enfin à permettre aux utilisateurs autorisés de créer, modifier ou supprimer des éléments.

Présentation

Pour les utilisateurs autorisés, cette action peut se dérouler dans la même page que la visualisation de la liste des actualités. Modifier, donc, le code de `01_list_actu`, dans le dossier `02b_actualites/arch_contenu`, comme suit :

```
<dtml-if "AUTHENTICATED_USER.hasRole(this, ['administration'])">
  <FORM ACTION="" METHOD="POST">
</dtml-if>
```

```
<table class="hztable" border="0" cellspacing="1" cellpadding="1">
  <tr class="hztable">
    <dtml-if "AUTHENTICATED_USER.hasRole(this, ['administration'])">
      <th class="hztable">Sélect.</th>
    </dtml-if>
    <th class="hztable">Date</th>
    <th class="hztable">Titre</th>
  </tr>

  <dtml-in "arch_actualites.objectValues(['Weballage Actu'])"
skip_unauthorized reverse sort="Date">
    <dtml-if sequence-odd>
      <dtml-call "REQUEST.set('table_class_', ' class=\x22hztable_odd\x22 ')">
    <dtml-else>
      <dtml-call "REQUEST.set('table_class_', ' class=\x22hztable_even\x22 ')">
    </dtml-if>
    <tr <dtml-var table_class_>>
      <dtml-if "AUTHENTICATED_USER.hasRole(this, ['administration'])">
        <td <dtml-var table_class_> align="center"><input type="checkbox"
           value="<dtml-var id>" name="actu_id:list"></td>
      </dtml-if>
      <td <dtml-var table_class_>><dtml-var "Date.strftime('%d/%m/%Y')"></td>
      <td <dtml-var table_class_>><A HREF="<dtml-var absolute_url>"><dtml-var
         Titre></A></td>
    </tr>
  </dtml-in>

</TABLE>

<P class="contenu">
<dtml-if "AUTHENTICATED_USER.hasRole(this, ['administration'])">
  Eléments sélectionnés :
  <input type="submit" name="proc_supprimer:method" value="Supprimer">
  </FORM>
</dtml-if>
     </P>
```

Nous avons simplement rajouté un test du rôle de l'utilisateur : s'il possède le rôle administration, le tableau devient un formulaire, avec une case à cocher pour sélectionner chaque actualité, et un bouton pour les supprimer.

Pour la création, nous allons placer un formulaire dans un autre `DMTL Document` du dossier `arch_contenu`, appelé `02_creation` :

```
<FORM action="" method="POST">

<table class="hztable" border="0" cellspacing="1" cellpadding="1">
  <tr class="hztable">
    <th class="hztable">Date</th>
    <td class="hztable"><INPUT type="text"
      name="DateFR:text:required"
      size="10"
      value="<dtml-var "_.DateTime().strftime('%d/%m/%Y')">"></
    td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Titre</th>
    <td class="hztable"><INPUT type="text"
      name="Titre:string:required"
      size="40"
      value=""></
    td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Résumé</th>
    <td class="hztable"><TEXTAREA
      name="Resume:text:required"
      wrap="soft"
      cols="20"
      rows="4"
      value=""></TEXTAREA></
    td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Contenu</th>
    <td class="hztable"><TEXTAREA
      name="Contenu:text:required"
      wrap="soft"
      cols="50"
      rows="20"
      value=""></TEXTAREA></
    td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Accès</th>
    <td class="hztable">
      <INPUT type="checkbox" name="ACCES:list" value="Anonymous"
       >Tout le monde<BR>
```

```
      <INPUT type="checkbox" name="ACCES:list" value="intranet">Intranet
       (salariés)<BR>
      <INPUT type="checkbox" name="ACCES:list" value="extranet">Extranet
      (clients)<BR>
    </td>
    </SELECT>
  </tr>
</TABLE>
<P class="contenu">
<input type="submit" name="proc_creer:method" value="Créer">
</P>

</FORM>
```

Ce code n'appelle pas de commentaire particulier : il s'agit simplement de la génération d'un formulaire en HTML. La page de traitement du formulaire sera un dossier appelé `proc_creer`.

Il ne reste plus qu'à créer le code correspondant à la création ou à la suppression d'un élément.

Traitements

Création d'une actualité

Le code de création d'une actualité doit se trouver dans la `DTML Method` `proc_init` du dossier `proc_créer`, lui-même situé sous `02b_creation`. On procède à l'instanciation de la ZClasse comme ceci :

```
# Création de l'objet et de ses propriétés
<dtml-with "arch_actualites.manage_addProduct['weballage_actu']
➥.weballage_actu_factory">
  <dtml-call "REQUEST.set('actu_id', '%s%s' % (AUTHENTICATED_USER,
  ➥_.DateTime().strftime('%Y%m%d%H%M%S')))">
  <dtml-with "weballage_actu.createInObjectManager(actu_id, REQUEST)">
      <dtml-call "propertysheets.proprietes.manage_changeProperties(REQUEST)">
      <dtml-call "manage_permission('View', roles=ACCES, acquire=0)">
      <dtml-call "manage_permission('Access contents information', roles=ACCES,
       acquire=0)">
    </dtml-with>
</dtml-with>
```

Remarque

L'utilisation d'un `<dtml-call>` *pour appeler* `proc_init` *procure un avantage indéniable : tout texte en dehors du DTML est ignoré. Il est donc possible de documenter le code en saisissant simplement les commentaires entre deux balises DTML. Ici, nous avons fait précéder le commentaire du signe # pour qu'il ressorte mieux, mais ce n'est pas une obligation.*

Ce code crée une actualité dans le dossier `arch_actualites` : observez le premier `<dtml-with>`, qui utilise l'acquisition pour obtenir la factory de notre ZClasse dans le contexte de `arch_actualites`.

Dès que la ZClasse est instanciée, nous changeons ses propriétés avec `manage_changeProperties`, puis ses droits avec `manage_permission` (pour les permissions `View` et `Access contents information`). Pour que ce code fonctionne, il faut bien veiller à ce que les noms des champs dans le formulaire `02_creation` correspondent aux noms des propriétés de la ZClasse. De même, il faut veiller à ce que les noms des rôles intranet, extranet et Anonymous soient corrects dans les balises `<INPUT type="checkbox">`.

Ce code pose tout de même un problème : en effet, nous souhaitons que dans tous les cas les droits soient tout de même accordés aux rôles administration et Manager, même si l'utilisateur ne coche aucune case. Pour y parvenir, il suffit de rajouter ce code au début de la `DTML Method` :

```
# Permissions par défaut : Manager & administration
<dtml-if ACCES>
  <dtml-call "ACCES.append('Manager')">
  <dtml-call "ACCES.append('administration')">
<dtml-else>
  <dtml-call "REQUEST.set('ACCES', ['Manager', 'administration'])">
</dtml-if>
```

Conversion de la date

Un dernier problème, dans le code précédent, a trait à la gestion de la date : la date est entrée en français par les utilisateurs, mais la méthode `_.DateTime` ne sait traiter que des dates au format américain (jour et mois inversés).

Il nous faut donc effectuer nous-mêmes la conversion : créer dans `proc_créer` un script `Python` appelé `meth_frDateTime` et qui prend un seul paramètre `date`. On y entre le code suivant :

```
import string

date = string.strip(date)
if len(date) != 10:
  raise "Date invalide : La date doit être au format jj/mm/aaaa", date

try:
  (jour, mois, annee) = map(lambda x: int(x), string.split(date, '/'))
except:
  raise "Date invalide : La date doit être au format jj/mm/aaaa", date

return DateTime(annee, mois, jour)
```

Si le script n'arrive pas à convertir la date, une exception est renvoyée. Ajouter maintenant ces deux lignes au début de `proc_init` :

```
# Calcul de la date
<dtml-call "REQUEST.set('Date', meth_frDateTime(DateFR))">
```

Redirection

Tel qu'il est, ce code peut fonctionner (à condition de l'exécuter avec le rôle Manager, pour les raisons que nous allons détailler) et, lorsque l'objet est créé, la liste, mise à jour, s'affiche. Mais si l'utilisateur clique sur le bouton Recharger de son navigateur, un deuxième objet est créé ! En effet, nous sommes toujours dans la page de traitement du formulaire et, lorsque l'utilisateur clique sur Recharger, les paramètres passés dans la requête `POST` sont rechargés en même temps que la page.

La manière la plus simple d'éviter cet inconvénient est de rediriger le navigateur sur une page – sans données de formulaire –, sitôt l'actualité créée. Pour cela, rajouter à la fin de la méthode `proc_init` du dossier `proc_créer` les lignes suivantes :

```
# Retour à la liste des actualités
<dtml-call "RESPONSE.redirect(_['02b_actualites'].absolute_url())">
```

Annuaire

L'étape suivante consiste à créer l'annuaire des utilisateurs. Nous avons vu que chaque entrée de l'annuaire devait contenir les éléments suivants :

- nom ;
- prénom ;
- site ;
- numéro de téléphone ;
- zone de texte libre (en structured text).

ZClasse `weballage_fiche`

Voici la procédure qu'il faut suivre pour créer la ZClasse avec laquelle on va gérer les entrées de l'annuaire.

1. Créer un produit `weballage_annuaire` (on aurait pu également créer un produit commun regroupant `weballage_actu`).
2. Créer, dans ce produit, une ZClasse `weballage_fiche` (de méta-type `Weballage Fiche`) représentant une entrée d'annuaire. Comme nous souhaitons qu'il soit possible de recher-

cher des enregistrements, dérivez votre classe de `CatalogAware`. Comme toujours, n'oubliez pas de cocher la case « Include Zope persistant object base classes ».

3. Créer, pour cette classe, une feuille de propriétés appelée `fiche`, et contenant les propriétés suivantes :

Nom	Type	Valeur
nom	String	
prenom	String	
telephone	String	
site	Selection	ListeSites
email	String	
contenu	Text	

Voici la feuille de propriétés qui en résulte (voir figure 15-11) :

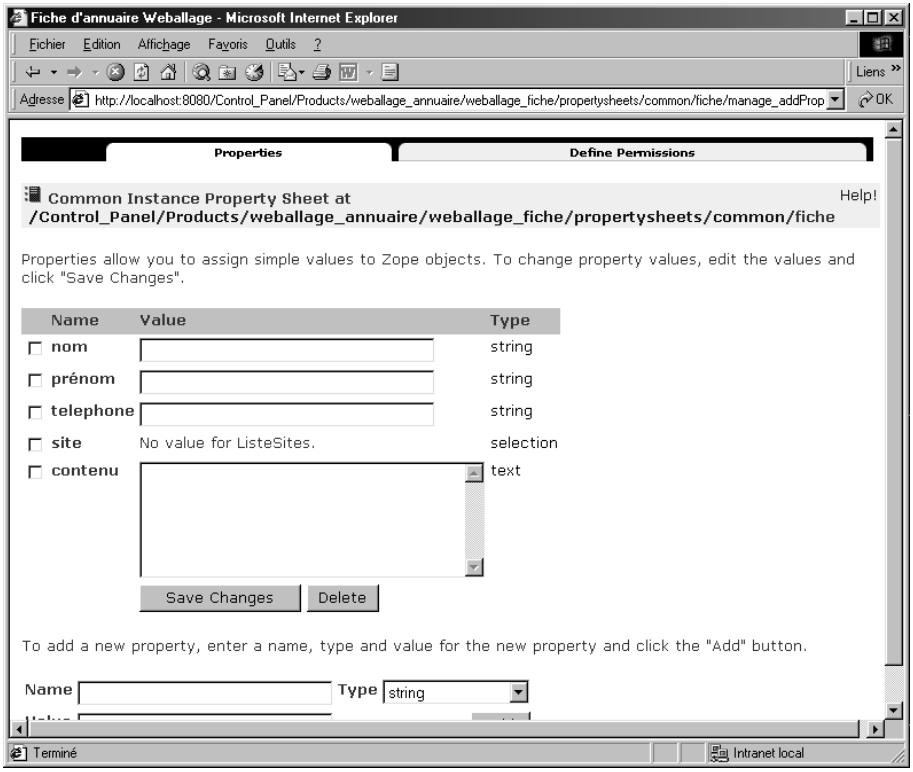


Figure 15-11
Feuille de propriétés `fiche`

Nous utilisons une variable `ListeSites` pour sélectionner le site de la fiche qui n'existe pas encore. Nous verrons plus bas comment la créer.

4. Créer une vue `Propriétés` associée à la fiche de propriétés ainsi créée.

Notre ZClasse est créée – mais nous aurons l'occasion d'y revenir, notamment pour modifier ses méthodes de création.

ZClasse `weballage_annuaire`

Dans l'arborescence de Zope, toutes les fiches peuvent être conservées au même endroit. Idéalement, elles devraient être contenues dans un dossier qui renferme un objet `Z Catalog` et une propriété `ListeSites` pour la sélection du site. En revanche, aucun autre type d'objet que des entrées d'annuaire ne pourrait être créé dans ce dossier. Le mieux, pour y parvenir, est de créer une nouvelle ZClasse, `weballage_annuaire` (méta-type), dérivant `ObjectManager` (voir figure 15-12).

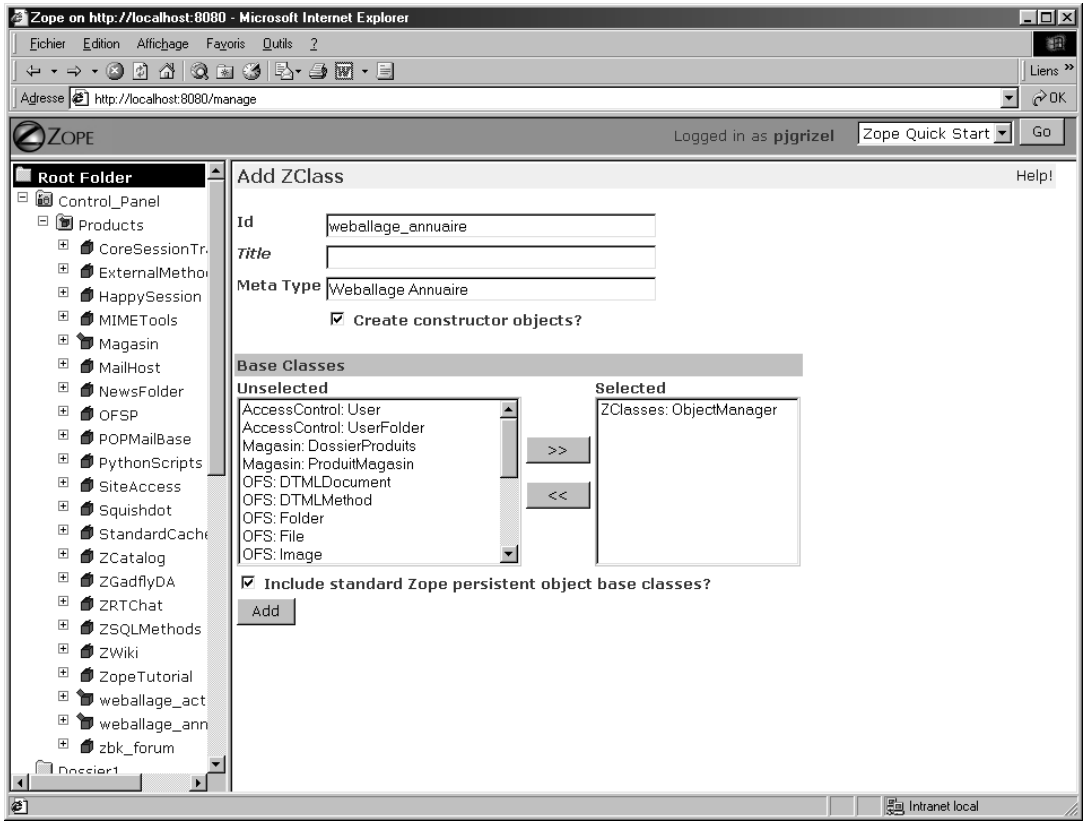


Figure 15-12

Création de la ZClasse `weballage_annuaire`

Pour indiquer à la ZClasse `weballage_annuaire` qu'elle ne peut contenir que des objets de type `Weballage Fiche`, cliquer sur l'onglet Subobjects et ne sélectionner que le type `Weballage Fiche` (voir figure 15-13).

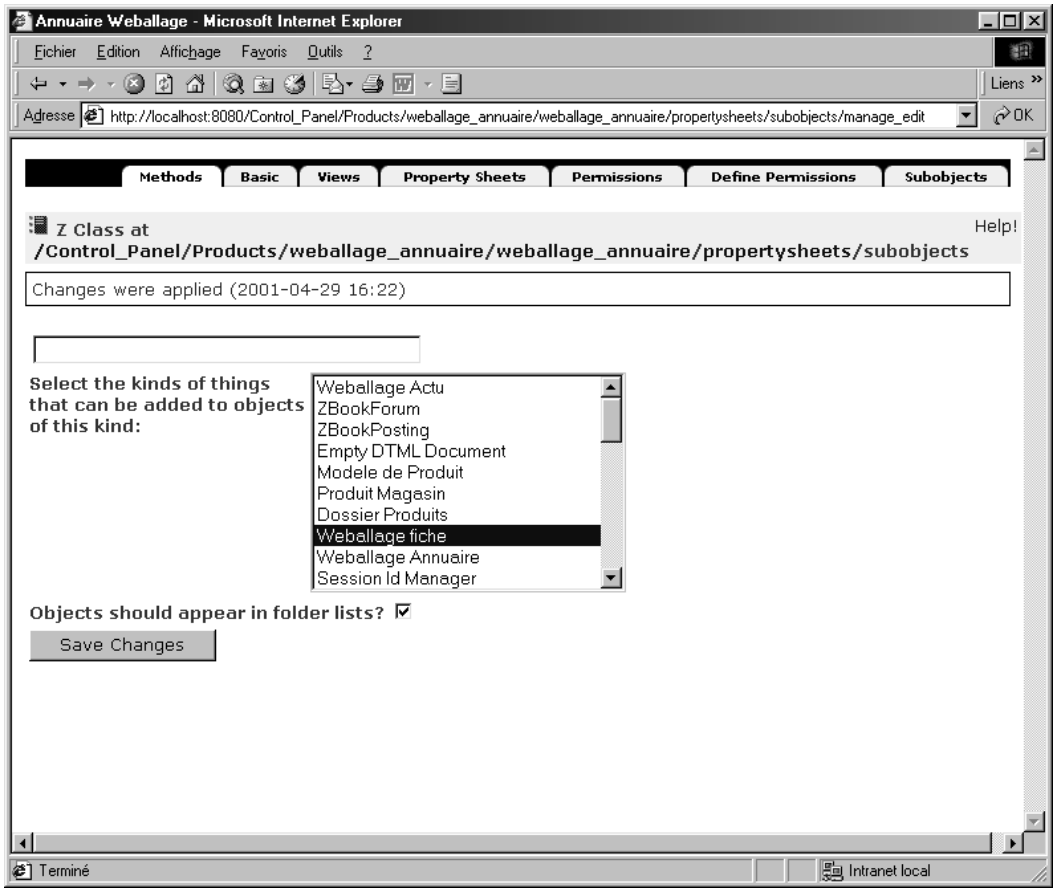


Figure 15-13
Sélection du type de sous-objets

On peut maintenant créer une feuille de propriétés Annuaire, et une propriété `ListeSites` de type `Lines` qui contienne la liste des sites possibles, séparés par des sauts de lignes.

Créer une vue Propriétés associée à la fiche de propriétés ainsi définie. Créer également une vue Sécurité associée à la méthode `manage_access`.

Puis, pour chaque objet annuaire, il faut un ZCatalog, appelé `Catalog` : il servira à cataloguer toutes les entrées créées dans cet annuaire. Les méta-données de ce catalogue sont `nom`, `prenom`, `site`, `email` et `telephone`. La propriété `contenu`, en revanche, ne se prête pas à l'utilisation en tant que méta-donnée. Le catalogue nécessite les index suivants :

Nom	Type
`nom`	*TextIndex*
`prenom`	*TextIndex*
`site`	*FieldIndex*
`contenu`	*TextIndex*
`email`	*TextIndex*

Ici, l'indexation sur le champ `téléphone` n'a pas grand intérêt. En revanche, l'indexation sur le contenu de la fiche personnelle peut être intéressante pour permettre des recherches dans le corps des fiches.

Ce catalogue doit être créé au moment de la création d'un objet de type `weballage_annuaire`, c'est-à-dire dans la méthode `weballage_annuaire_add`. Effacez la méthode `weballage_annuaire_add` créée par défaut et remplacez-la par un script Python prenant en paramètres `id` et `REQUEST`, et contenant le code suivant :

```
  # Création de l'objet weballage_annuaire
  obj = context.weballage_annuaire.createInObjectManager(id, {})

  # Création du catalogue
  obj.manage_addProduct['ZCatalog'].manage_addZCatalog(id="Catalog",
itle="Catalogue de l'annuaire")

  # Définition des métadonnées
  obj.Catalog.manage_addColumn('nom')
  obj.Catalog.manage_addColumn('prenom')
  obj.Catalog.manage_addColumn('site')
  obj.Catalog.manage_addColumn('telephone')
  obj.Catalog.manage_addColumn('email')

  # Définition des index
  obj.Catalog.manage_addIndex('nom', 'TextIndex')
  obj.Catalog.manage_addIndex('prenom', 'TextIndex')
  obj.Catalog.manage_addIndex('site', 'FieldIndex')
  obj.Catalog.manage_addIndex('email', 'TextIndex')
  obj.Catalog.manage_addIndex('contenu', 'TextIndex')

  # Redirection
```

```
try:
    u=context.DestinationURL()
except:
    u=REQUEST['URL2']
     REQUEST.RESPONSE.redirect(u+'/manage_workspace')
```

Ce code permet de créer l'objet `Catalog` et de positionner ses options d'index et de métadonnées en regard des champs que nous venons de définir.

Intégration `weballage_fiche` *et* `weballage_annuaire`

Les classes `weballage_fiche` et `weballage_annuaire` sont certes créées, mais elles doivent coopérer étroitement pour rendre le service désiré.

Réindexation

Le seul point important de l'intégration concerne le catalogue : nous avons vu que, lorsqu'un objet dérivé de `CatalogAware` est créé, il est indexé dans le premier catalogue acquis portant l'`id catalogue`. En revanche, lorsqu'un objet est modifié, il n'est *plus* mis à jour dans le catalogue : nous allons donc écrire un script Python `ModifierFiche` permettant de modifier la fiche et de mettre à jour automatiquement le catalogue. Il doit prendre les paramètres (dans l'ordre) `nom`, `prenom`, `telephone`, `email`, `site`, `contenu` et contenir le code suivant :

```
context.propertysheets.ProprietesProduit.manage_changeProperties(
  {
    "nom": nom,
    "prenom": prenom,
    "site": site,
    "telephone": telephone,
    "contenu": contenu,
    "email": email,
  }
)
context.reindex_object()
```

La méthode `reindex_object`, dérivée de `CatalogAware`, permet de mettre à jour les index de l'objet.

Pour que cette méthode soit appelée dans la vue Propriétés, il faudrait créer dans les méthodes de la ZClasse une DTML Method `frm_modifierProprietes` affichant un formulaire similaire à celui de la feuille de propriétés, mais appelant `ModifierFiche`. Comme nous allons écrire un tel formulaire dans le cadre du site, nous ne le décrivons pas ici.

Page et méthode de création

Les objets de type `Weballage Fiche` ne permettent pas pour l'instant de spécifier des propriétés lors de leur création. Il faut donc modifier les méthodes `weballage_fiche_add` et `weballage`

_fiche_addForm situées dans le produit weballage_annuaire pour prendre en compte les six propriétés de la classe. Modifier le formulaire comme suit :

```
<HTML>
<HEAD><TITLE>Add Weballage Fiche</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#000099" VLINK="#555555">
<H2>Add Weballage fiche</H2>
<form action="weballage_fiche_add"><table>
<tr>
    <th>Nom</th>
    <td><input type="text" name="nom:required"></td>
</tr>
<tr>
    <th>Prénom</th>
    <td><input type="text" name="prenom:required"></td>
</tr>
<tr>
    <th>Email</th>
    <td><input type="text" name="email:required"></td>
</tr>
<tr>
    <th>Site</th>
    <td><select name="site:required">
      <dtml-in ListeSites>
        <option value="&dtml-sequence-item;">&dtml-sequence-item;</option>
      </dtml-in>
    </select></td>
</tr>
<tr>
    <th>Numéro de téléphone</th>
    <td><input type="text" name="telephone"></td>
</tr>
<tr>
    <th>Informations personnelles</th>
    <td><textarea cols="30" rows="10" name="contenu"></textarea></td>
</tr>
<tr><td></td><td><input type=submit value=" Add "></td></tr>
</table></form>
</body></html>
```

On doit détruire la méthode weballage_fiche_add pour créer un script Python portant le même nom, et prenant en paramètres nom, prenom, telephone, site, email, contenu, REQUEST et redirection=1. Ici, contrairement aux actualités pour lesquelles nous avions dû créer un id unique pour chaque actualité, nous allons utiliser le champ email. Si d'aventure l'adresse e-mail de l'utilisateur était modifiée par la suite, la propriété email serait mise à jour alors que la propriété id resterait la même – mais cela n'a pas grande importance – compte tenu, d'une

part, de ce que les adresses e-mail ne sont jamais recyclées et que, d'autre part, le champ `id` n'est jamais utilisé explicitement.

Le paramètre `redirection` permet, s'il est explicitement passé en tant que valeur « fausse », d'empêcher la méthode de création de rediriger le navigateur vers le dossier contenant la fiche d'annuaire : en effet, dans le site définitif, l'utilisateur devra être redirigé vers une autre URL – nous verrons cela par la suite.

Voici le corps du script :

```
obj = context.weballage_fiche.createInObjectManager(email, {})
email = email + "@weballage.com"
obj.ModifierFiche(nom, prenom, telephone, email, site, contenu)
if redirection:
    try:
        u=context.DestinationURL()
    except:
        u=REQUEST['URL2']
    REQUEST.RESPONSE.redirect(u+'/manage_workspace')
```

La partie purement logique est terminée : il ne reste plus qu'à écrire l'interface qui va permettre de créer, consulter et modifier des entrées d'annuaire.

Interface de consultation

Avant toute chose, créer un dossier `04_annuaire` au même niveau que les autres parties du site si ce n'est pas déjà fait. Dans `04_annuaire`, créer un objet `Weballage Annuaire` et lui donner l'identifiant `annuaire`. Dans cet annuaire, sélectionner l'onglet Propriétés et entrer quelques lignes dans la zone ListeSites. On peut alors, sous l'environnement de développement de Zope, créer quelques objets dans `annuaire` pour s'assurer que les méthodes que l'on vient de créer se comportent correctement.

Dans `04_annuaire`, créer une méthode DTML appelée `html_liste_fiches`, et contenant le code suivant :

```
<table class="hztable">
  <tr>
    <th class="hztable">Nom</th>
    <th class="hztable">Prénom</th>
    <th class="hztable">Site</th>
    <th class="hztable">Téléphone</th>
    <th class="hztable">Email</th>
  </tr>
```

```
  <dtml-in "annuaire.Catalog.searchResults(REQUEST)" sort="nom">
    <dtml-if sequence-odd>
      <dtml-call "REQUEST.set('table_class_', ' class=\x22hztable_odd\x22 ')">
    <dtml-else>
      <dtml-call "REQUEST.set('table_class_', ' class=\x22hztable_even\x22 ')">
    </dtml-if>

    <tr <dtml-var table_class_>>
      <td <dtml-var table_class_>>
        <dtml-var nom>
      </td>
      <td <dtml-var table_class_>>
        <dtml-var prenom>
      </td>
      <td <dtml-var table_class_>>
        <dtml-var site>
      </td>
      <td <dtml-var table_class_>>
        <dtml-var telephone>
      </td>
      <td <dtml-var table_class_>>
        <a href="mailto:<dtml-var email>"><dtml-var email></a>
      </td>
    </tr>
  </dtml-in>
</table>
```

Nous verrons ultérieurement tout l'intérêt de placer ce code dans une méthode plutôt que dans un document. Créer ensuite un dossier `arch_contenu`. Puis y créer un DTML Document appelé `01_consultation` permettant d'afficher sous forme de liste triée alphabétiquement les entrées de l'annuaire. Entrer simplement le code suivant :

```
<dtml-var html_liste_fiches>
```

On peut se placer sur l'URL *http://localhost:8080/weballage.com/04_annuaire* pour observer le résultat (voir figure 15-14) :

Pour l'interface de consultation d'une fiche, nous allons nous baser sur les propriétés de l'acquisition. Pour en comprendre le fonctionnement, créer la fiche d'un utilisateur *toto* (pour ce faire, saisir *toto* dans la zone `email`). Pour l'instant, si on tente d'accéder à l'URL *http://localhost:8080/weballage.com/04_annuaire/toto*, la page qui liste les fiches apparaît à l'écran : en effet, la méthode `index_html` étant rendue par défaut par Zope et cette méthode n'existant *pas* pour l'objet de type `Weballage Fiche`, Zope retourne la méthode `index_html` acquise.

Pour afficher une fiche, il suffit alors de créer un objet pour lequel `index_html` correspond à l'apparence d'une page de visualisation d'une fiche, et de faire en sorte que cet objet acquière lui-même la fiche à afficher.

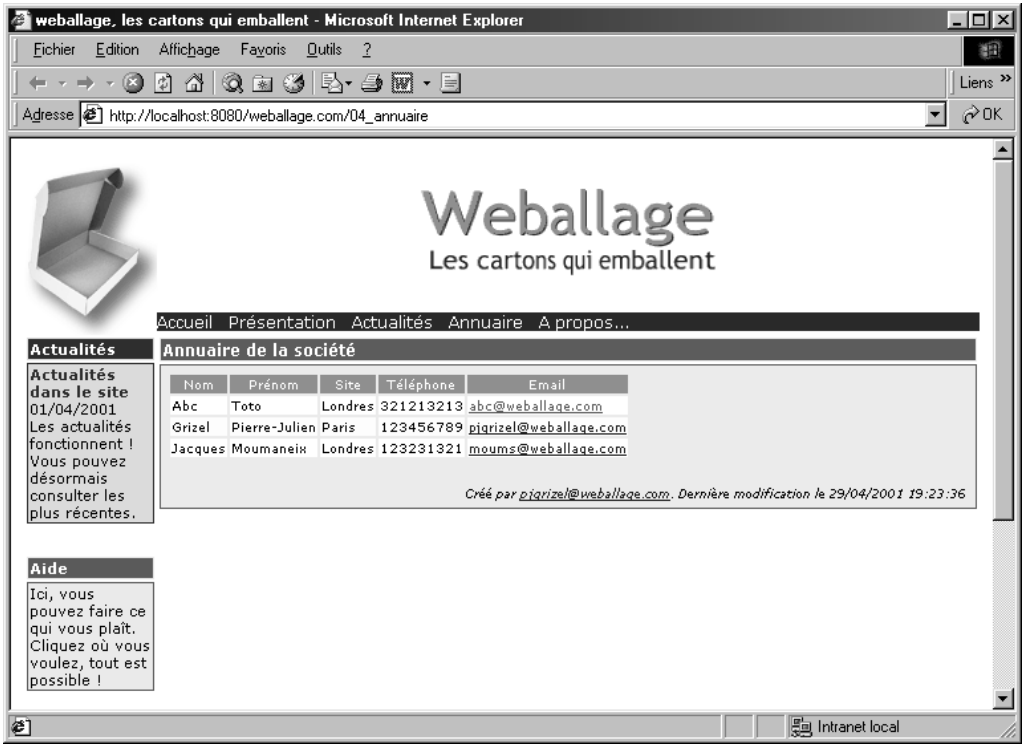


Figure 15-14
Affichage de l'annuaire

Voici comment procéder :

1. Créer, dans le dossier `04_annuaire`, un dossier appelé `voir_fiche`.
2. Dans `voir_fiche`, créer un dossier `arch_contenu`, conformément à l'architecture documentaire que nous avons établie.
3. Créer, dans `arch_contenu`, un document DTML `01_Fiche` contenant le code suivant :

```
<table class="hztable">
  <tr class="hztable">
    <th class="hztable">Nom</th>
    <td class="hztable"><dtml-var nom></td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Prénom</th>
    <td class="hztable"><dtml-var prenom></td>
  </tr>
  <tr class="hztable">
```

```
    <th class="hztable">Site</th>
    <td class="hztable"><dtml-var site></td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Téléphone</th>
    <td class="hztable"><dtml-var telephone></td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Email</th>
    <td class="hztable">
      <a href="mailto:<dtml-var email>"><dtml-var email></a>
    </td>
  </tr>
</table>
```

4. Créer un document DTML `02_PagePerso` contenant simplement le code suivant :

```
<dtml-var contenu fmt="structured-text">
```

On peut maintenant accéder à l'URL *http://localhost:8080/weballage.com/04_annuaire/voir_fiche/toto* pour constater que la fiche est rendue correctement (figure 15-15) :

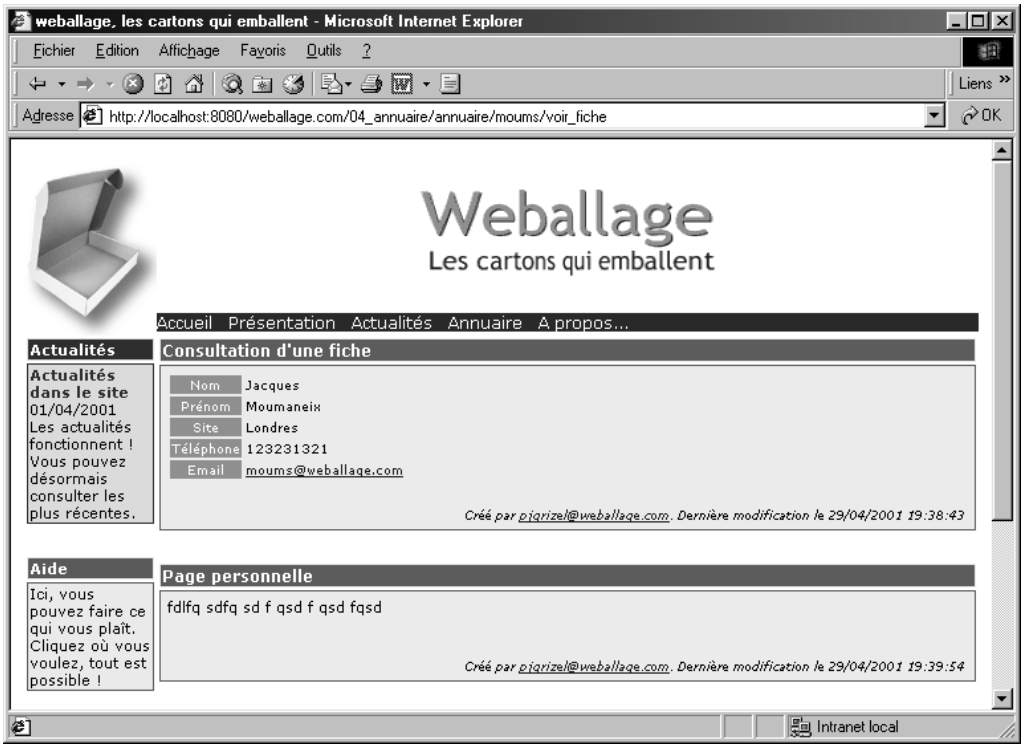


Figure 15-15
Affichage d'une fiche

Pour permettre aux utilisateurs de cliquer sur une entrée dans l'annuaire, il faut modifier, dans le document `01_consultation`, la ligne

```
        <dtml-var nom>
```

de cette façon

```
        <a href="<dtml-var getURL>/voir_fiche"><dtml-var nom></a>
```

ce qui a pour effet de transformer la zone Nom du tableau d'affichage des entrées de l'annuaire en un lien vers la fiche correspondante.

Interface de création et / ou de modification

Globalement, une interface de modification ou de création d'une entrée d'annuaire a, dans l'un ou l'autre cas, la même structure – sauf que, dans le cas de la modification, les valeurs existantes doivent être affichées dans le formulaire.

Créer, dans `04_annuaire`, un dossier `editer_fiche`, dans lequel on crée ensuite un autre folder `arch_contenu`. Puis on y crée un document DTML `01_Edition` contenant le code suivant :

```
<dtml-if nom>
  <form action="proc_modifier">
<dtml-else>
  <form action="proc_creer">
</dtml-if>

<table class="hztable">
  <tr class="hztable">
    <th class="hztable">Nom</th>
    <td class="hztable">
      <input type="text"
      name="nom:required"
      value="<dtml-var nom missing>">
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Prénom</th>
    <td class="hztable">
      <input type="text"
      name="prenom:required"
      value="<dtml-var prenom missing>">
    </td>
  </tr>

  <tr class="hztable">
```

```
    <th class="hztable">Site</th>
    <td class="hztable">
      <select name="site:required">
        <dtml-in "annuaire.ListeSites">
          <option value="<dtml-var sequence-item>"><dtml-var
sequence-item></option>
        </dtml-in>
      </select
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Téléphone</th>
    <td class="hztable">
      <input type="text"
      name="telephone:required"
      value="<dtml-var telephone missing>">
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Adresse email</th>
    <td class="hztable">
      <input type="text"
      name="email:required"
      value="<dtml-var email missing>">@weballage.com
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Informations personnelles</th>
    <td class="hztable">
      <textarea name="contenu:text" cols="50" rows="20"><dtml-var
      ➥contenu missing></textarea>
    </td>
  </tr>

</table>

<dtml-if nom>
  <input type="submit" value="Modifier">
<dtml-else>
  <input type="submit" value="Créer">
</dtml-if>
```

Ce formulaire n'appelle pas de commentaire particulier, excepté que la balise `<dtml-if>` est utilisée au début et à la fin du document pour tester si l'on est en en phase de création ou en

modification. Si la variable `nom` peut être acquise, alors on doit modifier une fiche ; sinon, on doit la créer.

Le code de modification est pris en charge par un script Python, placé dans le dossier `editer_fiche`, appelé `proc_modifier`, et prenant les paramètres `REQUEST`, `RESPONSE`, `nom`, `prenom`, `telephone`, `email`, `site` et `contenu` :

```
# Recherche l'objet Fiche
for obj in REQUEST.PARENTS:
  if obj.meta_type == "Weballage Fiche":
    fiche = obj
    break

# Modifie la fiche
fiche.ModifierFiche(nom, prenom, telephone, email, site, contenu)

# Redirige
RESPONSE.redirect(REQUEST.URL2)
```

La boucle, au début, permet de s'assurer que la méthode `ModiferFiche` est appelée sur une fiche et pas sur un autre type d'objet.

Créer, de la même manière, un script `proc_creer`, prenant les mêmes paramètres que `proc_modifier`, et contenant le code suivant :

```
# Crée la fiche
fiche =
context.annuaire.manage_addProduct['weballage_annuaire'].weballage_fiche.
createInObjectManager(email, REQUEST)

# Configure les informations de la fiche
email = email + "@weballage.com"
fiche.ModifierFiche(nom, prenom, telephone, email, site, contenu)

# Redirige
RESPONSE.redirect(REQUEST.URL2)
```

On peut tester la création et la modification respectivement avec les URL *http://localhost:8080/weballage.com/04_annuaire/annuaire/editer_fiche* et *http://localhost:8080/weballage.com/04_annuaire/annuaire/voir_fiche/toto/editer_fiche*. Ici, aucun paramètre n'est passé dans la requête ; toutes les informations sont contenues dans l'URL : l'acquisition est extrêmement pratique dans ce contexte.

Il faut maintenant créer les boutons respectifs qui permettent de créer et de modifier les entrées d'annuaire. Le bouton de création doit être affiché dans la page de visualisation de l'annuaire et ne doit apparaître que pour les administrateurs. À la fin du document `01_consulatation`, rajouter les lignes mises en évidence :

```
<dtml-var html_liste_fiches>

<dtml-if "_.SecurityCheckPermission('Add Weballage Fiches', annuaire)">
  <form action="editer_fich">
    <input type="submit" value="Créer une fiche">
  </form>
</dtml-if>
```

Si l'utilisateur a la permission d'ajouter une fiche dans l'annuaire, le bouton apparaît ; sinon, il n'apparaît pas.

De même, dans le document `voir_fiche/arch_contenu/01_Fiche`, ajouter la partie en gras pour permettre l'affichage du bouton :

```
<table class="hztable">
  <tr class="hztable">
    <th class="hztable">Nom</th>
    <td class="hztable"><dtml-var nom></td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Prénom</th>
    <td class="hztable"><dtml-var prenom></td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Site</th>
    <td class="hztable"><dtml-var site></td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Téléphone</th>
    <td class="hztable"><dtml-var telephone></td>
  </tr>
  <tr class="hztable">
    <th class="hztable">Email</th>
    <td class="hztable">
      <a href="mailto:<dtml-var email>"><dtml-var email></a>
    </td>
  </tr>
</table>

<dtml-in PARENTS>
  <dtml-if "meta_type == 'Weballage Fiche'">
    <dtml-if "_.SecurityCheckPermission('Manage properties', _
     ['sequence-item'])">
      <form action="editer_fiche">
        <input type="submit" value="Modifier la fiche">
      </form>
    </dtml-if>
  </dtml-if>
</dtml-in>
```

L'annuaire fonctionne désormais : il est possible de lire les entrées, de les créer et de les modifier. Ce principe peut être aisément appliqué à la création d'un formulaire de suppression.

Interface de recherche

Afficher l'annuaire tel quel présente un intérêt limité. Il peut être intéressant de proposer un formulaire de recherche pour sélectionner les utilisateurs à afficher. Créer le formulaire suivant dans un document DTML appelé `00_recherche` et situé dans `04_annuaire/arch_contenu` :

```
Rechercher...
<form action="recherche" method="get">
  <table class="hztable">
    <tr class="hztable">
      <td class="hztable">
        <select name="champ">
          <option value=""></option>
          <option value="nom">Nom égal à...</option>
          <option value="nom">Prénom égal à...</option>
          <option value="nom">Adresse email...</option>
        </select>
      </td>
      <td class="hztable">
        <input type="text" name="valeur" value="">
      </td>
    </tr>

    <tr class="hztable">
      <td class="hztable">Sur le site...</td>
      <td class="hztable">
        <select name="site">
          <option value=""></option>
          <dtml-in "annuaire.ListeSites">
            <option value="<dtml-var sequence-item>">
            ➥<dtml-var sequence-item></option>
          </dtml-in>
        </select>
      </td>
    </tr>

    <tr class="hztable">
      <td class="hztable">Contenant le texte...</td>
      <td class="hztable">
        <input type="text" name="contenu" value="">
      </td>
  </table>

  <input type="submit" value="Rechercher...">

</form>
```

Créer ensuite un dossier `recherche` dans `04_annuaire`, puis créer le dossier `arch_contenu` et, à l'intérieur, un document `01_resultats` contenant le code suivant :

```
<dtml-if champ>
  <dtml-if valeur>
    <dtml-call "REQUEST.set(champ, valeur)">
  </dtml-if>
</dtml-if>

<dtml-var html_liste_fiches>
```

Et voilà ! Aussitôt, l'interface de recherche est parfaitement fonctionnelle. Voici la page de recherche (voir figure 15-16) :

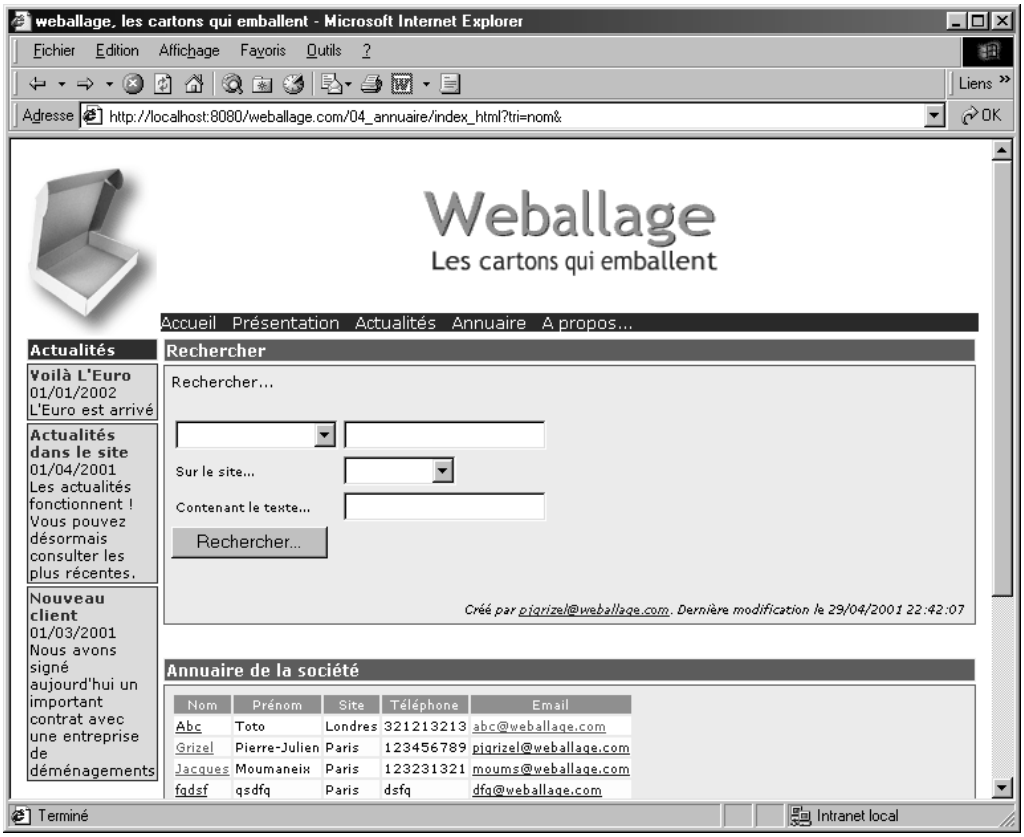


Figure 15-16
Page de recherche

Et voici la page d'affichage des résultats (voir figure 15-17) :

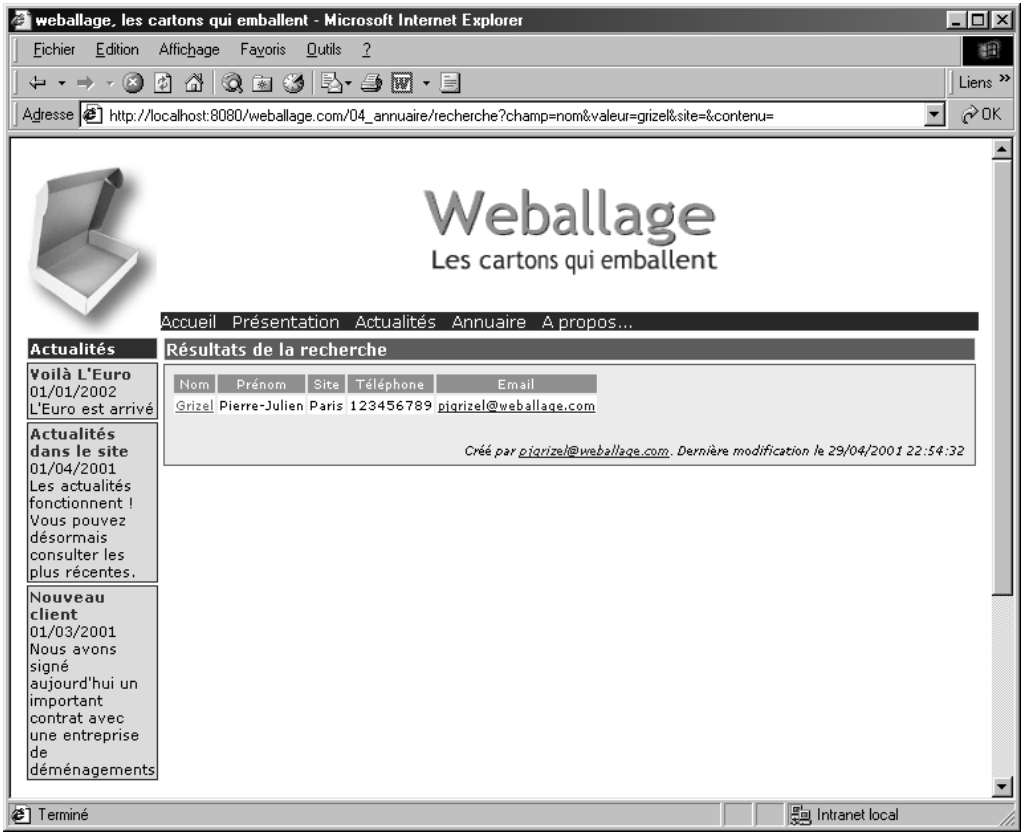


Figure 15-17
Page d'affichage des résultats de la recherche

Sécurité et gestion des utilisateurs

La méthode que nous avons utilisée pour créer l'interface de gestion de l'annuaire est telle que, pour réserver l'accès à l'annuaire aux membres de l'intranet, il suffit de changer la sécurité de l'objet `04_annuaire`, en empêchant l'acquisition des permissions `View` et `Access Content Information`, et en les affectant au rôle intranet.

Néanmoins, il subsiste un problème. Si vous effectuez vos tests avec le rôle Manager, vous n'aurez probablement pas remarqué qu'un utilisateur ne peut pas modifier sa propre fiche… En fait, les fiches utilisateur ne sont pas réellement associées à un utilisateur du site. Mieux

vaudrait créer, avec la fiche l'utilisateur dans le dossier `acl_users`, et le rendre propriétaire de sa propre fiche. Pour cela, il faut tout d'abord modifier le formulaire de création comme suit :

```
<dtml-if nom>
  <form action="proc_modifier">
<dtml-else>
  <form action="proc_creer">
</dtml-if>

<table class="hztable">
  <tr class="hztable">
    <th class="hztable">Nom</th>
    <td class="hztable">
      <input type="text"
      name="nom:required"
      value="<dtml-var nom missing>">
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Prénom</th>
    <td class="hztable">
      <input type="text"
      name="prenom:required"
      value="<dtml-var prenom missing>">
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Site</th>
    <td class="hztable">
      <select name="site:required">
        <dtml-in "annuaire.ListeSites">
          <option value="<dtml-var sequence-item>"><dtml-var
sequence-item></option>
        </dtml-in>
      </select
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Téléphone</th>
    <td class="hztable">
      <input type="text"
      name="telephone:required"
      value="<dtml-var telephone missing>">
    </td>
  </tr>
```

```
  <tr class="hztable">
    <th class="hztable">Adresse email</th>
    <td class="hztable">
      <input type="text"
      name="email:required"
      value="<dtml-var email missing>">
      <dtml-unless email>@weballage.com</dtml-unless>
    </td>
  </tr>

  <tr class="hztable">
    <th class="hztable">Informations personnelles</th>
    <td class="hztable">
      <textarea name="contenu:text" cols="50" rows="20"><dtml-var
contenu missing></textarea>
    </td>
  </tr>

  <dtml-unless>
    <!-- Création d'un utilisateur -->

    <tr class="hztable">
      <th class="hztable">Créer un compte utilisateur</th>
      <td class="hztable">
        <input type="checkbox" name="creer_compte">
      </td>
    </tr>

    <tr class="hztable">
      <th class="hztable">Mot de passe</th>
      <td class="hztable">
        <input type="text" name="password">
      </td>
    </tr>

    <tr class="hztable">
      <th class="hztable">Confirmation du mot de passe</th>
      <td class="hztable">
        <input type="text" name="confirm">
      </td>
    </tr>

  </dtml-unless>

</table>

<dtml-if nom>
  <input type="submit" value="Modifier">
<dtml-else>
  <input type="submit" value="Créer">
</dtml-if>
```

Il faut également modifier le script `proc_creer`. Rajouter les paramètres `password=""`, `confirmation=""` et `creer_compte=0` à la fin de la liste, et modifier le code comme suit :

```
# Crée la fiche
fiche = context.annuaire.manage_addProduct['weballage_annuaire'].weballage
➥_fiche.createInObjectManager(email, REQUEST)

# Configure les informations de la fiche
name = email
email = email + "@weballage.com"
fiche.ModifierFiche(nom, prenom, telephone, email, site, contenu)

# Configure la sécurité
if creer_compte:
    context.acl_users.manage_users(
      'Add',
      {
        "name": name,
        "password": password,
        "confirm": confirm,
        "roles": ['intranet'],
        "domains": [],
      })
    fiche.manage_addLocalRoles(name, ["Owner"])

# Redirige
RESPONSE.redirect(REQUEST.URL2)
```

La méthode `acl_users.manage_users` permet de créer le compte utilisateur avec les paramètres spécifiés. Puis, une fois le compte créé, on définit le rôle local Owner sur la fiche. Il n'est pas possible de « donner » la propriété d'un objet à un utilisateur : c'est la raison pour laquelle le rôle Owner est utilisé.

Dans l'onglet Sécurité de l'objet `annuaire`, sélectionner le rôle Owner en regard de la permission `Manage properties` : c'est ce rôle qui permettra à l'utilisateur de modifier son propre objet par la suite.

On peut désormais valider la création d'utilisateurs avec cette méthode : le site permet désormais d'ajouter des utilisateurs à l'intranet d'une de ses pages.

En résumé…

Dans cette première étude de cas, nous avons suivi les étapes qui jalonnent la construction d'un site riche et puissant.

En premier lieu, il ne faut pas négliger l'étape de la conception : savoir où seront stockées les données, choisir les outils (internes à Zope ou externes) qui vont permettre de résoudre les différents problèmes. Ensuite, l'intégration de l'architecture sous-jacente doit se faire le plus naturellement possible, avec l'adjonction progressive de méthodes et de documents prenant en charge l'apparence générale du site : une fois cette étape effectuée, le programmeur n'a plus à se soucier de la présentation.

Dès le départ, il convient aussi d'établir les règles de gestion des utilisateurs (rôles, permissions).

Nous avons étudié pas à pas la création du module de gestion des actualités, au travers de ZClasses. L'utilisation conjointe des fonctionnalités simples de Zope et des méthodes de l'API permet d'aboutir rapidement à la fonctionnalité désirée.

Pour la gestion de l'annuaire, nous avons créé deux ZClasses imbriquées, ainsi qu'un catalogue. Ici encore, l'API de Zope est d'une grande utilité lorsqu'il s'agit de configurer précisément le comportement des objets.

L'intranet de `weballage` pourrait être encore étendu pour prendre en charge d'autres fonctionnalités… Mais nous allons nous pencher, dans l'étude de cas suivante, sur le problème du commerce électronique : présenter un catalogue de produits, permettre aux utilisateurs anonymes de commander ces produits et aux responsables du site de suivre l'évolution de ces commandes.