14

Stratégies de conception avec Zope

Si tu ne sais pas où tu vas, tu ne risques pas d'y arriver
(proverbe arabe).

Même si nous avons pris soin, tout au long de cet ouvrage, de présenter différentes techniques qui permettent de bien l'appréhender, Zope reste un outil complexe qui demande de la pratique pour être maîtrisé. Ce chapitre donne quelques pistes pour aider le concepteur à en tirer le meilleur parti. Nous aborderons ainsi deux aspects déterminants pour la conception d'un site en Zope, à savoir la sécurité et les considérations d'architecture, liées notamment à la séparation entre présentation et logique.

Sécurité

La sécurité est primordiale sous Zope. Elle repose sur l'utilisation de quatre éléments :

- **les utilisateurs** (au travers des dossiers `acl_users`) ;
- **les rôles** (au travers des dossiers `acl_users` ou de l'option Local roles de l'onglet Security) ;
- **les permissions** (avec l'onglet Security) ;
- **la propriété** (au moyen de l'onglet Ownership).

Tout au long de cet ouvrage, nous avons utilisé ces notions, mais il convient d'y revenir brièvement à la lumière de ce que nous avons appris sur Zope, pour mettre en évidence certains de ses comportements.

Authentification et acquisition

À travers l'exemple suivant, nous allons détailler concrètement comment les utilisateurs, associés à des rôles et aux permissions correspondantes, s'authentifient et comment intervient le mécanisme d'acquisition dans les procédures d'authentification.

Observons tout d'abord l'arborescence Zope de cette figure (voir figure 14-1) :

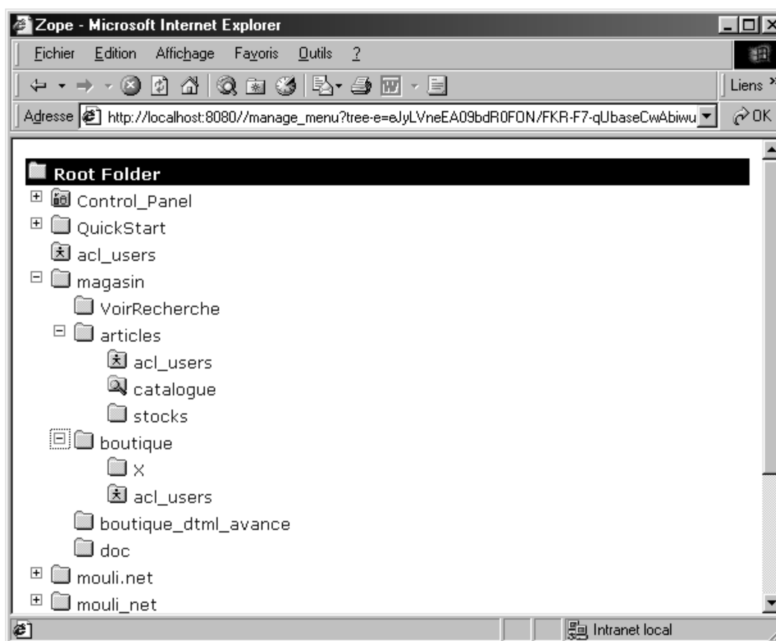

Figure 14-1
Acquisition des `acl_users`

On distingue nettement trois dossiers `acl_users`. Voici maintenant la liste des utilisateurs et des mots de passe définis pour ces trois `acl_users` :

`User Folder`	**Utilisateur**	**Mot de passe**
/acl_users	*pierre*	*pierre*
/acl_users	*paul*	*paul*
/magasin/articles/acl_users	*pierre*	*toto*

© Éditions Eyrolles, 2001

User Folder	Utilisateur	Mot de passe
/magasin/articles/acl_users	*paul*	*paul*
/magasin/boutique/acl_users	*jacques*	*xyz*
/magasin/boutique/acl_users	*pierre*	*tata*

Tous les utilisateurs ont le rôle Manager.

1. Créer un `DTML Document` appelé `request`, que l'on place à la racine du site et qui contient simplement :

```
<dtml-var REQUEST>
```

2. Créer, toujours dans la racine, un document `request_protege`, contenant lui aussi `<dtml-var REQUEST>`.
3. Dans l'onglet Security, changer les permissions de `request_protege` pour qu'il ne puisse être vu que par les utilisateurs dotés du rôle Manager (la case Acquire permission settings n'est donc pas cochée pour cette permission).

Lorsqu'un utilisateur se connecte à l'URL *http://localhost:8080/request_protege*, la fenêtre de demande de mot de passe s'affiche et l'utilisateur doit s'authentifier (voir figure 14-2).

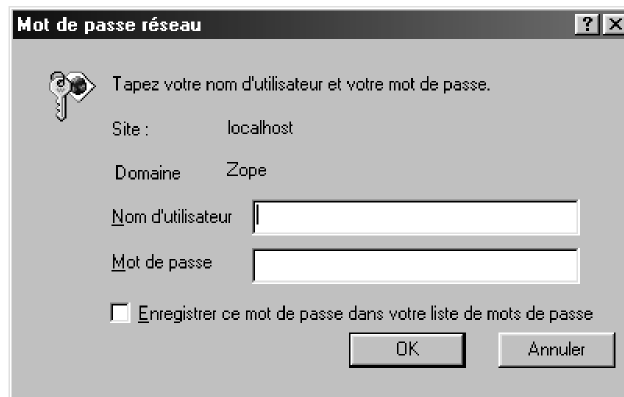

Figure 14-2
Fenêtre d'authentification du navigateur

Le seul dossier utilisé pour cette authentification est `/acl_users` ; autrement dit, seuls les utilisateurs *pierre* et *paul* peuvent se connecter au site, en utilisant respectivement les mots de passe *pierre* et *paul*.

Le contenu de la variable `REQUEST` s'affiche alors à l'écran et l'on peut observer dans la section Environ ces deux informations (si *pierre* se connecte) :

```
AUTHENTICATED_USER    pierre
AUTHENTICATION_PATH   '
```

© Éditions Eyrolles, 2001

La variable AUTHENTICATED_USER contient l'objet utilisateur authentifié. La variable AUTHENTICATION_PATH contient pour sa part le chemin dans lequel se situe le dossier acl_users qui a permis l'authentification.

> **Attention**
>
> *La variable* AUTHENTICATED_USER *contient un objet et non une chaîne de caractères (contrairement à* AUTHENTICATION_PATH*). Il est possible d'appliquer des méthodes et de lire des attributs de l'objet* AUTHENTICATED_USER*, comme nous le verrons ultérieurement.*

1. Fermer le navigateur, puis se connecter à l'URL *http://localhost:8080/magasin/articles/manage* pour contraindre Zope à vous identifier. Dans la fenêtre d'authentification, on entre pierre et toto.
2. Puis, dans la même fenêtre du navigateur, accéder à l'URL *http://localhost:8080/magasin/articles/request* et observer les deux variables :

```
AUTHENTICATED_USER        pierre
AUTHENTICATION_PATH       '/magasin/articles'
```

La variable AUTHENTICATION_PATH indique que l'utilisateur est authentifié grâce au dossier /magasin/articles/acl_users. Si on tente maintenant d'accéder à l'URL *http://localhost:8080/magasin/articles/request_protege*, la fenêtre d'authentification apparaît à nouveau et on ne peut accéder au document que si on s'authentifie avec un utilisateur et un mot de passe appartenant au dossier /acl_users.

L'authentification est toujours *relative* à un dossier acl_users. Elle est alors acquise pour le reste du site, mais, pour cette raison, si on souhaite accéder à une ressource protégée par un autre acl_users, on doit à nouveau s'authentifier. Ici, le document request_protege est protégé par le dossier /acl_users tandis que l'utilisateur pierre était authentifié à partir du dossier /magasin/articles/acl_users.

Si on ferme à nouveau le navigateur et que l'on refait exactement la même manipulation, mais en utilisant *pierre/pierre* comme utilisateur et mot de passe, le rendu de l'URL *http://localhost:8080/magasin/articles/request* affichera les informations suivantes :

```
AUTHENTICATED_USER       pierre
AUTHENTICATION_PATH      ''
```

On voit ici clairement ce qui s'est passé : la saisie du mot de passe *pierre* a empêché le dossier /magasin/articles/acl_users de valider l'authentification de l'utilisateur *pierre*. La demande d'authentification a donc été transmise à un dossier situé plus haut hiérarchiquement, soit /acl_users dans notre cas. Et l'accès à l'URL *http://magasin/articles/request_protege* n'entraîne pas une nouvelle demande de mot de passe.

© Éditions Eyrolles, 2001

Si on ferme à nouveau notre navigateur pour renouveler l'expérience, cette fois avec *paul* (il possède le même mot de passe dans les deux dossiers `acl_users`), voici ce qu'indique le document *http://localhost:8080/magasin/articles/request* :

```
AUTHENTICATED_USER      paul
AUTHENTICATION_PATH      '/magasin/articles'
```

L'accès à l'URL *http://localhost:8080/magasin/articles/request_protege* n'entraîne pas l'apparition d'une nouvelle boîte de dialogue de demande de mot de passe mais renvoie les informations suivantes :

```
AUTHENTICATED_USER      paul
AUTHENTICATION_PATH      ''
```

Ici, la variable `AUTHENTICATION_PATH` a été automatiquement modifiée pour prendre en compte le dossier `acl_users` situé à la racine.

Cet exemple montre les mécanismes d'authentification dans Zope et met en évidence les conclusions suivantes :

- L'authentification d'un utilisateur est toujours relative à un dossier `acl_users`. Zope tente d'authentifier un utilisateur en acquérant le dossier `acl_users` le plus proche. S'il n'y parvient pas, il acquiert le suivant, et ainsi de suite, jusqu'à la racine.
- Les permissions affectées aux rôles d'un objet sont toujours relatives au premier dossier `acl_users` acquis par l'objet.
- Pour qu'un utilisateur obtienne une permission sur un objet, il faut qu'il soit doté d'un des rôles affectés à cette permission *dans le contexte du dossier `acl_users` présent ou acquis par l'objet*. Dans notre premier exemple, un utilisateur authentifié en tant que Manager dans `/magasin/articles` n'avait pas accès à la méthode `request_protege` qui nécessite le rôle Manager dans le contexte de `/`.

Ces règles sont très importantes dans Zope, tout en étant assez intuitives. Le cas échéant, il est toujours utile de bien avoir en tête leur formulation.

Remarque

Les exemples présentés ici n'ont d'autre objet que de montrer le fonctionnement de l'authentification sous Zope. D'une manière générale, il est fortement déconseillé de créer les mêmes noms d'utilisateur dans plusieurs dossiers `acl_users`. Si un utilisateur doit posséder des rôles différents à travers le site, utiliser la fonctionnalité Local Roles de Zope.

Authentification HTTP

Un autre élément est mis en évidence au travers des exemples précédents. Dans le dernier cas, *paul* possède le même mot de passe pour `/magasin/articles` et `/`. Zope semble « se souvenir » du mot de passe de *paul*, puisqu'il n'affiche pas la fenêtre de saisie de mot de passe lors de l'appel du document `request_protege`.

© Éditions Eyrolles, 2001

Pourtant, il n'en est rien : Zope conserve les mots de passe sous forme cryptée et ne les décrypte jamais. Lorsqu'un utilisateur saisit son mot de passe dans la fenêtre d'authentification, Zope crypte ce mot de passe et le compare au mot de passe conservé sous forme *cryptée* : ce sont deux mots de passe cryptés qui sont comparés, et jamais deux mots de passe en clair. On ne peut, avec l'algorithme utilisé par Zope (SHA), *décrypter* le mot de passe d'un utilisateur, même en analysant le contenu de la base ZODB octet par octet.

Comment se fait-il, alors, que Zope semble se souvenir du mot de passe entre les différentes pages du site, et surtout au moment où l'authentification n'est plus gérée par le même dossier `acl_users` ?

La réponse est simple : le mot de passe n'est pas conservé par Zope mais par… le navigateur. L'authentification sous Zope fonctionne d'après les spécifications du protocole HTTP. La première fois qu'un utilisateur tente d'accéder à une page protégée, voici le dialogue qui se produit entre le navigateur et le serveur :

- Le navigateur contacte le serveur pour accéder à cette page.
- Le serveur retourne une erreur du type « non autorisé » au navigateur.
- Le navigateur affiche la fenêtre d'authentification à l'utilisateur.
- Une fois le nom d'utilisateur et le mot de passe entrés par l'utilisateur, le navigateur réitère sa requête auprès du serveur, en mentionnant les informations d'authentification saisies par l'utilisateur dans sa requête.
- Si les informations sont correctes, le serveur retourne la page.
- Le navigateur conserve alors en mémoire le nom d'utilisateur et le mot de passe.

Voici maintenant ce qui se passe lorsque l'utilisateur tente d'accéder à nouveau à la même page, alors qu'il a déjà entré son nom d'utilisateur et son mot de passe valides :

- Le navigateur contacte le serveur pour accéder à cette page.
- Le serveur retourne une erreur du type « non autorisé » au navigateur[1].
- Le navigateur sait qu'il a en mémoire les informations d'authentification concernant le serveur : il réitère sa requête auprès du serveur, en mentionnant les informations d'authentification conservées en mémoire.
- Si les informations sont correctes, le serveur retourne la page.

Dans le cas où les informations d'authentification ne sont pas correctes (par exemple, dans notre premier exemple où l'utilisateur *pierre* n'a pas le même mot de passe pour `/` ou pour `/magasin/articles`), le navigateur affiche à nouveau la fenêtre d'authentification et conserve le nom d'utilisateur et le mot de passe pour ce serveur, conformément à l'URL passée par l'utilisateur.

1. Certains navigateurs prennent l'initiative d'omettre ces deux premières étapes lorsqu'un utilisateur accède à une URL pour laquelle un jeton existe.

© Éditions Eyrolles, 2001

> **Remarque**
> *Les informations d'authentification sont toujours transmises « en clair » par le navigateur au serveur (elles sont codées mais pas cryptées). C'est la raison pour laquelle le protocole HTTP n'est pas sécurisé.*

Ces informations stockées en mémoire par le navigateur et relatives à un serveur, on les appelle jeton d'authentification. Un jeton peut être relatif à tout un site (c'est le cas pour *paul*) ou juste à une branche de l'arborescence d'un site (c'est le cas pour *pierre*, qui, dans le dernier exemple, possède deux jetons : l'un pour `localhost:8080/` et l'autre pour `localhost:8080/articles/magasin/`). Un jeton relatif à un répertoire est utilisé pour tous ses sous-répertoires – tant qu'une erreur d'authentification n'est pas soulevée par le serveur. Le répertoire et les sous-répertoires concernés par un jeton sont déterminés par le serveur, et correspondent à la valeur de `AUTHENTICATION_PATH`.

Un jeton n'est perdu que dans l'un des cas suivants :

- la fenêtre portant le jeton est fermée (Internet Explorer uniquement) ;
- l'utilisateur quitte complètement le navigateur ;
- le serveur indique une erreur d'authentification pour le jeton *et* l'utilisateur ne parvient pas à s'identifier à nouveau.

> **Remarque**
> *Sous Internet Explorer, les jetons sont portés par une fenêtre. Sous Netscape Navigator, ils sont portés par l'application elle-même et sont donc partagés entre toutes les fenêtres. C'est la raison pour laquelle il est très dangereux, sous Netscape, de se connecter en tant que Manager et de laisser son poste de travail sans surveillance – même en ayant fermé la fenêtre de Zope : il faut tout à fait quitter Netscape pour effacer le jeton.*

Le serveur n'a aucun moyen de demander explicitement au navigateur d'effacer un jeton. Il n'est donc pas possible de proposer, dans un site, un bouton Se déconnecter : il ne s'agit pas d'une limitation propre à Zope, mais au protocole HTTP lui-même.

En revanche, il existe un moyen facile de se « désauthentifier » : il suffit de déclencher une erreur d'authentification qui, elle-même, entraîne l'affichage de la fenêtre d'authentification du navigateur *et* d'entrer un nom d'utilisateur ou un mot de passe erroné (cliquer sur Annuler ne suffit pas) : le jeton est alors considéré comme étant non valide par le navigateur et il est effacé. On peut par exemple placer, à la racine de son site, une méthode `logout` contenant le code suivant :

```
<dtml-raise "Unauthorized">Vous êtes déconnecté</dtml-raise>
```

Pour se déconnecter, il suffit d'appeler l'URL *http://localhost:8080/logout*, d'entrer un mauvais mot de passe et de valider, puis de cliquer sur Annuler.

© Éditions Eyrolles, 2001

> **Remarque**
>
> *Il s'agit de la méthode utilisée par Zope pour le bouton Logout dans l'environnement de développement.*

Ce principe d'authentification explique pourquoi il n'est *pas* possible d'intercepter avec `standard_error_message` une erreur d'authentification (voir le chapitre 6 consacré au DTML avancé). En effet, la page d'erreur d'authentification de Zope (voir figure 14-3) est systématiquement renvoyée au navigateur, *pour chaque requête concernant une page protégée*, même si le navigateur possède un jeton (dans ce cas, elle n'apparaît pas aux yeux de l'utilisateur – mais elle est quand même retournée au navigateur avant la page demandée). Le mécanisme de renvoi de cette page est donc forcément différent de celui qui permet de rendre une page depuis Zope : la page d'erreur est écrite « en dur » dans le code de Zope et le seul moyen que l'on ait de la modifier est de changer le code source de Zope.

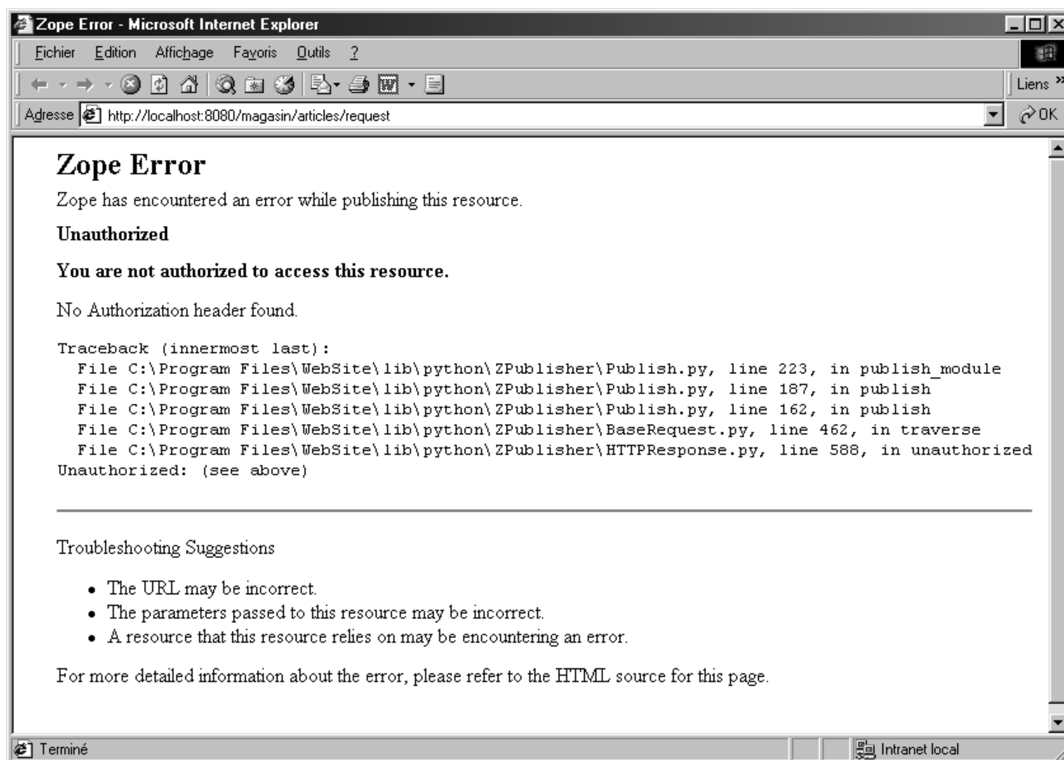

Figure 14-3
Page d'erreur d'authentification de Zope

© Éditions Eyrolles, 2001

Le mode de fonctionnement de l'authentification HTTP explique aussi certains comportements étranges qui se produisent avec SiteRoot. Un jeton est toujours relatif à un site, et SiteRoot permet de modifier l'adresse de base du site, vue par le navigateur. Comme ce dernier n'a aucun moyen à sa disposition qui lui permette d'enregistrer la « supercherie » de SiteRoot, il considère qu'il n'a pas de jeton pour le nouveau site et demande à nouveau l'authentification. C'est la raison pour laquelle la fenêtre d'authentification peut être affichée plusieurs fois lors de l'administration d'un site qui contient des objets SiteRoot.

Comment pallier l'authentification HTTP

Le protocole HTTP n'est pas sécurisé et pose un certain nombre de problèmes – dont celui de la déconnexion. Il existe des alternatives qui permettent de s'affranchir du protocole HTTP, qui nécessitent toutefois l'installation de produits Zope. Avec ces produits, il est par exemple possible de stocker le jeton d'authentification d'une manière cryptée et sécurisée dans un cookie présent sur le poste du client – puis d'effacer simplement ce cookie pour déconnecter l'utilisateur.

Le plus connu de ces produits alternatifs auxquels on peut recourir dans ce cas est `User Manager`, qui permet de configurer les mécanismes d'authentification d'une manière très fine, et de stocker la base des utilisateurs autre part que dans des objets `User` de Zope (par exemple, dans LDAP, dans une base SQL ou dans des ZClasses). User Manager peut être téléchargé depuis le site de Zope (*http://www.zope.org*) mais, en raison de sa puissance, il est recommandé de faire une lecture approfondie de la documentation qui y a trait.

Deux exceptions qui confirment la règle

Les mécanismes décrits ici sont valides pour tous les utilisateurs de Zope… excepté deux, situés chacun aux deux extrêmes : le superutilisateur et l'utilisateur anonyme.

Le superutilisateur

Le superutilisateur créé lors de l'installation de Zope n'existe dans aucun dossier `acl_users`. Il est stocké dans un fichier à part, pour permettre de dépanner Zope même si le dossier `acl_users` est corrompu (voir le chapitre 13 consacré à l'administration système de Zope pour de plus amples informations à ce sujet).

Aussi, les principes exposés dans ce chapitre au sujet de l'authentification par `acl_users` ne concernent-ils pas le superutilisateur – et c'est la raison pour laquelle il est conseillé de se servir le moins possible de cet utilisateur particulier.

En revanche, le superutilisateur est toujours authentifié avec le protocole HTTP décrit plus haut.

Utilisateur anonyme

Lorsqu'une permission est affectée au rôle Anonymous, les choses se déroulent d'une toute autre façon. L'utilisateur anonyme n'étant lié à aucun `acl_users`, lorsqu'un utilisateur accède à une

© Éditions Eyrolles, 2001

URL accessible à un utilisateur anonyme, le comportement de Zope est nettement plus direct. En effet, si des informations d'authentification ne sont pas fournies par le navigateur, Zope affecte l'objet `ANONYMOUS_USER` à la variable `AUTHENTICATED_USER` et rend l'objet au navigateur.

Dans la pratique, cette démarche peut avoir une conséquence : dans notre description du dialogue qui s'instaure entre navigateur et serveur, nous avons vu que le navigateur envoyait (ou du moins pouvait envoyer, car certains navigateurs ne passent pas par cette étape) une requête *sans* informations d'authentification, même s'il possède un jeton pour l'URL. Dans ce cas, le serveur ne renvoie pas d'erreur d'authentification, et, bien que l'utilisateur soit authentifié pour le reste du site, il est traité comme `ANONYMOUS_USER` pour une page sur laquelle il n'y a pas de restrictions. C'est un comportement de Zope dont il faut tenir compte : lorsqu'on vérifie l'identité d'un utilisateur au moyen de `AUTHENTICATED_USER`, il faut toujours vérifier que le navigateur a bien été contraint par le serveur à transmettre son jeton d'authentification.

Remarque

Certains navigateurs – dont les dernières versions d'Internet Explorer et Netscape Navigator – renvoient systématiquement le jeton d'authentification s'ils l'ont en mémoire. Dans ce cas, la variable `AUTHENTICATED_USER` *est positionnée correctement.*

La variable `AUTHENTICATED_USER`

La sécurité revêt une importance particulière sous Zope. Les mécanismes offerts par Zope pour assurer la sécurité sont de deux types :

- sécurité au niveau de l'architecture (droits et permissions accordés aux différents objets de l'arborescence Zope), ou *sécurité déclarative* ;
- sécurité au niveau de l'API (c'est-à-dire les fonctions et méthodes capables de vérifier la sécurité dans le site), ou *sécurité programmée*.

Nous n'avons rencontré au fil de cet ouvrage que la gestion de la sécurité déclarative. La sécurité programmée intervient lorsque le programmeur souhaite, au sein d'une méthode, vérifier par programmation les droits et permissions d'un utilisateur. La variable `AUTHENTICATED_USER` permet de le faire : il s'agit d'un objet qui donne des informations sur l'utilisateur authentifié.

L'objet `AUTHENTICATED_USER` est porteur des méthodes suivantes (voir tableau 14-1) :

Tableau 14-1. Méthodes de l'objet `AUTHENTICATED_USER`

Méthode	Description
`getUserName()`	*Retourne le nom de l'utilisateur sous forme de chaîne.*
`hasRole`*(objet*`,` *rôles*`)`	*Retourne vrai si l'utilisateur possède un des rôles passés dans le contexte de objet.*
`getRoles`*(objet*`)`	*Retourne une liste de chaînes représentant les rôles de l'utilisateur dans le contexte de objet.*

© Éditions Eyrolles, 2001

Par exemple, dans une méthode de `magasin`, pour vérifier si l'utilisateur possède le rôle Manager sur le dossier `articles`, on peut utiliser le code DTML suivant :

```
<dtml-if "AUTHENTICATED_USER.hasRole(articles, ['Manager'])">
    ...
</dtml-if>
```

> **Remarque**
> *Dans Zope, les rôles et permissions sont toujours exprimés sous forme de chaînes de caractères.*

Il s'avère parfois pratique d'avoir la possibilité de tester si une permission est accordée à un objet particulier : la méthode `SecurityCheckPermission` effectue ce test, y compris en DTML (elle est accessible dans l'espace de noms). Par exemple, pour tester si, dans le contexte courant, la permission `View` est accordée à l'objet `articles`, on peut écrire le code suivant :

```
<dtml-if "_.SecurityCheckPermission('View', articles)">
    ...
</dtml-if>
```

Nous verrons en études de cas dans la quatrième partie que cette méthode permet d'adapter le contenu d'une page rendue aux permissions accordées à l'utilisateur.

Procuration (proxies)

Dans certains cas, assez rares, il est nécessaire de contourner les mécanismes de sécurité de Zope pour donner à tous les utilisateurs des rôles particuliers sur certaines méthodes. Nous allons en présenter un exemple concret.

Il est souvent très agréable d'afficher sur un document le nom de l'utilisateur qui l'a créé – c'est-à-dire le nom du propriétaire du document. Pour cela, les objets Zope définissent une méthode `owner_info()` avec laquelle on peut récupérer les informations souhaitées. Voici, par exemple, comment écrire un script Python, `proprietaire`, qui retourne le nom du propriétaire d'un objet :

```
return context.owner_info()['id']
```

Et voici un document qui met en œuvre cette méthode :

```
<dtml-var standard_html_header>
<h2><dtml-var title_or_id></h2>
<p>
Le document <dtml-var id> appartient à <dtml-var proprietaire>.
</p>
<dtml-var standard_html_footer>
```

© Éditions Eyrolles, 2001

Hélas, la méthode `owner_info` n'est accessible qu'à un utilisateur doté du rôle Manager… Et si on essaie de visualiser le document alors que l'on n'est pas connecté en tant que Manager, Zope signale une erreur d'authentification.

La solution à ce problème consiste à indiquer que le script `proprietaire` doit s'exécuter avec le rôle Manager quel que soit l'utilisateur qui y recourt. On appelle une telle méthode une « méthode ayant procuration » (*proxy method* en anglais) ou méthode mandataire.

Les `DTML Document`, `DTML Method` et scripts Python sont dotés d'un onglet Proxy (voir figure 14-4) qui permet d'affecter à ces objets des droits qu'ils peuvent ne pas avoir. Cliquer sur l'onglet Proxy du script, sélectionner le rôle Manager et valider.

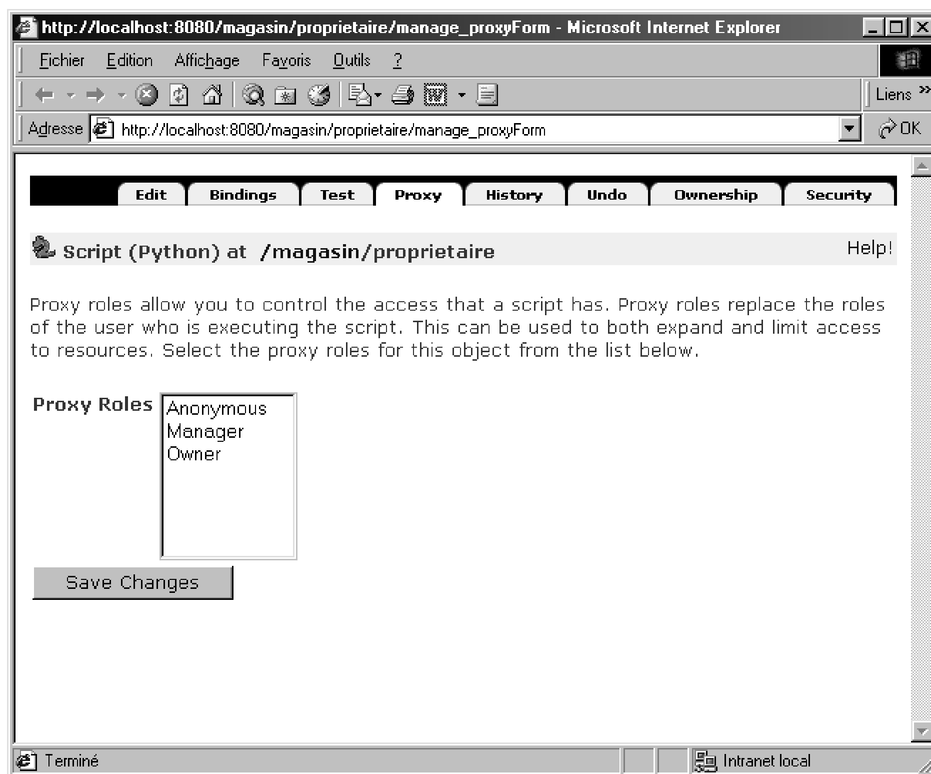

Figure 14-4
Onglet Proxy d'un objet

Une fois que cela est accompli, le script `proprietaire`, lorsqu'il est exécuté par quelque utilisateur que ce soit – y compris anonyme –, se comporte comme si l'utilisateur courant avait le rôle Manager et notre problème est résolu sans que l'on doive enfreindre les règles de sécurité de Zope.

© Éditions Eyrolles, 2001

> **Remarque**
>
> *Il n'est possible de donner un rôle mandataire à une méthode que si le propriétaire de la méthode (owner) possède lui-même ce rôle.*
>
> **Attention**
>
> *Les méthodes mandataires peuvent se révéler très dangereuses : il est très rare que l'on ait besoin d'y recourir, et le cas échéant, cela peut indiquer une erreur de conception dans la sécurité du site. En tout état de cause, ne donner de rôles mandataires qu'à un nombre très restreint de méthodes, et faire en sorte que le code de ces méthodes reste le plus petit possible pour limiter les risques d'effets de bord.*
>
> *Il est intéressant de noter qu'une méthode mandataire peut « gagner » en sécurité – comme dans notre exemple – ou « perdre » en sécurité : on peut notamment contraindre une méthode à s'exécuter avec le rôle Anonymous, bien que cela ne s'avère utile qu'en de rares cas.*

Bilan sur les permissions

Cette présentation des méthodes mandataires va nous permettre de passer en revue les conditions qui sont requises pour qu'un utilisateur ait le droit d'utiliser une méthode. Pour qu'un utilisateur obtienne une permission sur un objet, il faut que l'une des conditions suivantes soit remplie :

- La permission de l'objet doit être accordée au rôle Anonymous.
- La permission de l'objet doit être accordée à un rôle de l'utilisateur *et* elle doit aussi être accordée à un rôle du propriétaire de l'objet.
- L'objet doit être accédé depuis une méthode mandataire qui a un rôle pour lequel la permission est accordée *et* le propriétaire de la méthode mandataire doit posséder le rôle pour lequel la permission est accordée.
- Dans tous les cas, si un objet est accédé au travers d'une URL et que son `id` commence par `manage_`, il faudra obligatoirement posséder la permission `View Management Screens` pour accéder à cette page.
- Dans tous les cas, si une méthode ou un attribut commence par « _ », elle est *inaccessible* à partir d'un code DTML ou un script Python.

Dans les deux dernières règles, la double condition d'accès est impérative car il en résulte que des utilisateurs moins privilégiés ne peuvent obtenir des privilèges artificiellement. Le modèle de sécurité de Zope est éprouvé – et efficace.

Modules dans les scripts Python

Le principal reproche formulé à l'encontre des scripts Python concerne le peu de modules qui sont ainsi rendus disponibles pour le programmeur. Il est possible (bien que fortement déconseillé) de rendre disponibles d'autres modules dans tous les scripts Python de Zope. Il suffit d'utiliser les mécanismes de sécurité relatifs aux produits, en créant un nouveau produit.

© Éditions Eyrolles, 2001

Voici la marche à suivre.

1. Créer un répertoire (dont le nom importe peu) dans le répertoire `$ZOPE/lib/python/Products`. Par exemple, `$ZOPE/lib/python/Products/PythonScriptModules`.
2. Créer, dans ce répertoire, un fichier `__init__.py` contenant le code suivant :

   ```
   from Products.PythonScripts.Utility import allow_module
   allow_module(nom du module à autoriser)
   allow_module(nom du module à autoriser)
   ```

3. Créer autant de lignes `allow_module` que de modules à autoriser. Le nom du module doit être une chaîne de caractères. Par exemple, pour permettre l'importation du module de gestion des expressions rationnelles, écrire `allow_module('re')`.
4. Redémarrer Zope.

Les modules spécifiés sont alors accessibles depuis n'importe quel script Python. Il est désormais possible d'écrire, sans provoquer d'erreur :

```
import re
```

Attention

Cette technique peut s'avérer très dangereuse. Il faut éviter d'autoriser l'importation de modules qui manipulent des fichiers sur le disque, ou bien l'importation de modules inconnus ou peu, ou mal, documentés.

Sécurité dans les classes créées par le programmeur

Il arrive que ce soit dans un produit, ou dans une méthode externe, que l'on ait besoin de créer une méthode qui retourne une classe (voir l'étude de cas consacrée à LDAP, chapitre 17, par exemple). Zope, pour des raisons de sécurité, refuse alors que les utilisateurs accèdent aux attributs des objets instanciés à partir de cette classe.

Considérons une méthode externe contenant par exemple le code suivant :

```
def test(chaine):
    class X:
        def __init__(self, chaine):
            self.chaine = chaine

        def Contenu(self):
            return self.chaine

return X(chaine)
```

© Éditions Eyrolles, 2001

Et avec le code DTML suivant, qui fait appel à la méthode externe :

```
<dtml-var "test('Hello, World !').Contenu()">
```

Ce code provoquera une erreur d'authentification car il est interdit, sous Zope (en DTML ou dans un script Python), d'accéder à un objet qui n'est pas explicitement accessible.

Pour rendre l'objet accessible, il suffit de lui donner un attribut `__allow_acces_to _unprotected_subobjects__` qui vaut vrai. On peut donc par exemple modifier le fichier `.py` de la méthode externe comme suit :

```
def test(chaine):
    class X:
        __allow_access_to_unprotected_subobjects__ = 1
        def __init__(self, chaine):
            self.chaine = chaine

        def Contenu(self):
            return self.chaine

return X(chaine)
```

Le code s'exécute maintenant sans problème.

Architecture d'un site

L'architecture d'un site, c'est la façon dont les différents objets qui le composent sont organisés, et la manière dont ils interagissent entre eux. Définir l'architecture d'un site consiste à décrire avec précision le cheminement des données dans le serveur, à partir du clic d'un utilisateur jusqu'au renvoi de la page correspondante au navigateur.

Dans le modèle traditionnel du Web – qui implique le rendu de pages HTML statiques stockées sous forme de fichiers individuels –, cette architecture est réduite à sa plus simple expression.

> **Remarque**
>
> *Les schémas qui illustrent les différents modèles d'architecture utilisent la légende présentée ci-après (voir figure 14-5). Des numéros d'ordre sont indiqués sur les flèches afin de préciser la séquence d'appel des méthodes. Des lettres peuvent aussi indiquer différents cheminements possibles. Sauf cas exceptionnels, une page est toujours rendue au navigateur sur sa demande : ainsi les schémas débutent-ils généralement par une flèche « requête » provenant du navigateur et se terminent-ils par une flèche « réponse » qui y retourne.*

© Éditions Eyrolles, 2001

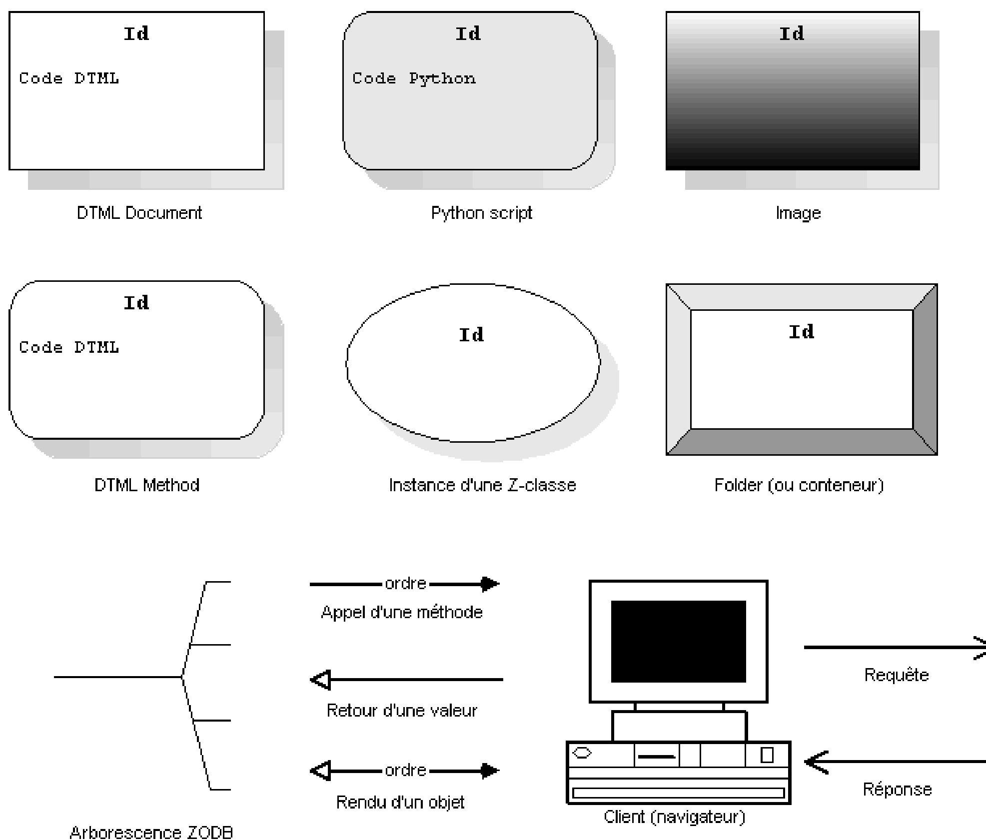

Figure 14-5

Conventions de description des modèles d'architecture.

Lors de l'importation d'un site vers Zope, ce dernier convertit les fichiers représentant des pages HTML en `DTML Document`. Le schéma suivant illustre le modèle traditionnel de rendu d'une page web (voir figure 14-6) :

© Éditions Eyrolles, 2001

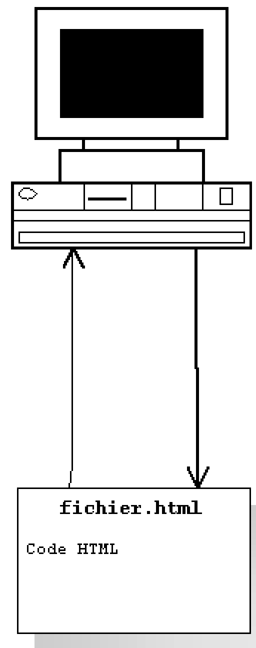

Figure 14-6
Le modèle traditionnel du Web

Nécessité d'une bonne architecture

Les risques du Web spaghetti dans Zope

Le modèle traditionnel statique posait un certain nombre de problèmes, en particulier celui de la redondance du code HTML. Dans une architecture de type statique en effet, l'homogénéité du site n'était obtenue qu'au prix de la répétition, dans chaque page, de toute la mécanique HTML – et ce en raison du fait que le langage HTML ne donne pas la possibilité d'incorporer un document HTML dans un autre document HTML (cela aurait pourtant épargné bien des soucis aux pionniers du Web). Tout l'intérêt d'un système de pages web dynamiques tient à ce que la tâche du programmeur en est allégée d'autant et qu'il n'a pas à répéter du code HTML dans chacune de ses pages.

Lors de l'écriture d'un site web dynamique, néanmoins, bien d'autres facteurs sont à prendre en compte, et il existe d'autres niveaux de réutilisation possibles. Par exemple, plusieurs formulaires, situés à des endroits différents du site peuvent faire appel à un même traitement, tout comme des pages différentes peuvent utiliser des données qui sont toujours générées de la même façon.

Comme pour tout projet, il est dangereux de se lancer tête baissée dans l'écriture d'un site sans avoir, au préalable, pensé à la façon dont les différents objets seront stockés dans l'arborescence de Zope. L'acquisition, si elle résout bien des problèmes, en pose d'autres. Il est très

© Éditions Eyrolles, 2001

facile d'obtenir rapidement un site dont la structure ressemble plus à un plat de spaghettis qu'à une structure organisée.

Le modèle implicite de Zope

Pourquoi ne pas reprendre l'architecture suggérée par Zope à savoir créer tout simplement des `DTML Document` commençant par `<dtml-var standard_html_header>` et s'achevant par `<dtml-var standard_html_footer>` (voir figure 14-7) ? Lors du traitement d'un formulaire, il suffit de créer un script Python ou une méthode DTML qui contient le code correspondant, et le tour est joué… La logique de présentation HTML est alors pour l'essentiel contenue dans `standard_html_header` et `standard_html_footer`.

Certes, nul besoin de réécrire dans chaque page le code HTML des en-têtes et pieds de page standard. Mais il faut pourtant bien réinsérer à chaque fois l'instruction d'insertion. Supposons un instant que notre site soit terminé et mis en production. Les utilisateurs vont commencer à saisir eux-mêmes des pages et, forcément, quelqu'un aura tôt ou tard recours à la maintenance car il aura oublié de mettre `<dtml-var standard_html_header>` au début de son `DTML Document`, ou `<dtml-var standard_html_footer>` à la fin. Cela signifie également que

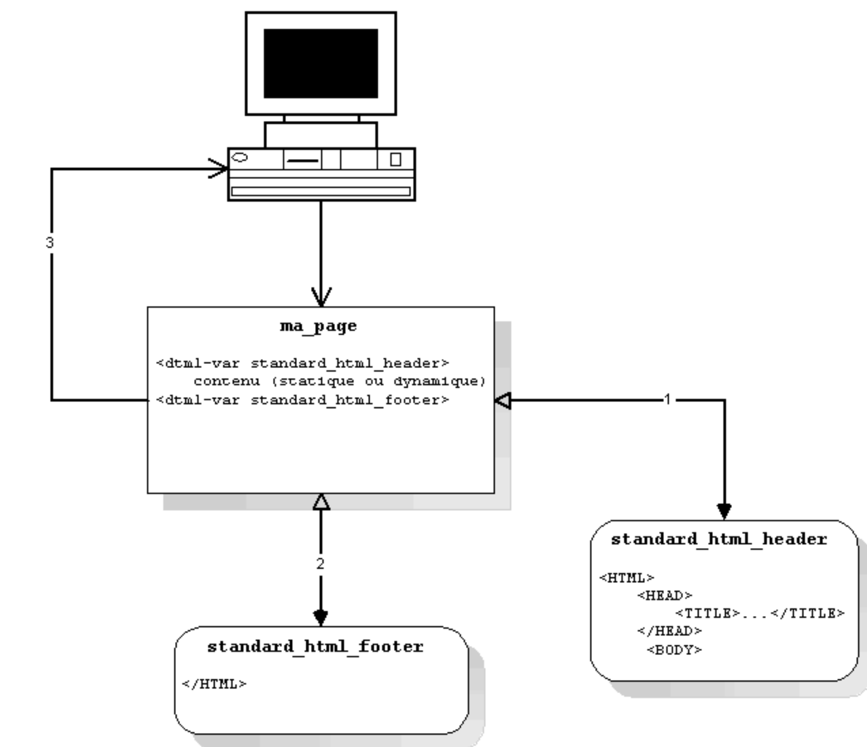

Figure 14-7
Architecture basée sur un `DTML Document`

© Éditions Eyrolles, 2001

toutes les pages visibles contiennent `<dtml-var standard_html_header>` et `<dtml-var standard_html_footer>` : alors que Zope propose au développeur et aux utilisateurs les moyens de s'affranchir de la redondance induite par le développement web, l'architecture proposée en standard dans Zope y fait massivement appel… Voilà qui est paradoxal – et surtout dissuasif.

Ce modèle présente un autre inconvénient – et non des moindres : il n'offre pas de véritable séparation entre le contenant et le contenu. Supposons que nous ayons besoin de proposer un bouton Page imprimable qui transforme la page HTML en une page de texte brut – plus facile à imprimer. Pour que les deux pages aient exactement le même contenu mais une présentation différente, il faut modifier `standard_html_header` et `standard_html_footer`. Comme ces deux documents sont inclus dans le corps du `DTML Document`, la seule solution pour modifier l'apparence de la page est de jouer avec l'acquisition. Mais en observant attentivement le schéma de la figure 14-8, on se rend compte que ce n'est pas possible dans la pratique.

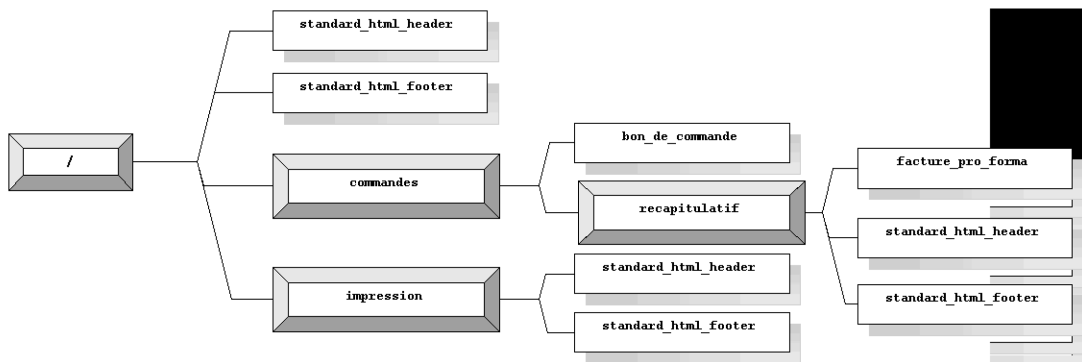

Figure 14-8
Page imprimable au travers de l'acquisition

Dans cet exemple, pour accéder au bon de commande, on utilise l'URI `/commandes/bon_de_commande`. Pour la version imprimable du bon de commande, on utilise l'URI `/commandes/bon_de_commande/impression`, qui « surcharge » par acquisition `standard_html_header`. Supposons que l'affichage ou l'impression d'un modèle de facture pro forma se fasse selon une autre disposition, c'est-à-dire avec d'autres versions de `standard_html_header/footer` : l'impression doit elle aussi être différente. Par conséquent, il faudrait , dans le schéma, rajouter un dossier `impression` contenant `standard_html_header/footer`, et ainsi faire en sorte qu'une page utilise un autre modèle que le modèle « par défaut » à chaque fois !

La solution à ce problème tient en un meilleur découpage d'une page web, en éléments qui puissent être aisément surchargés.

© Éditions Eyrolles, 2001

Présentation et traitement : deux architectures complémentaires

Présentation et traitement

Pour tenter de définir ce qu'est une bonne architecture de site, nous allons distinguer deux parties :

1. l'architecture de présentation ;
2. l'architecture de traitement.

La première permet de décrire la manière dont le code HTML d'une page est construit. La seconde décrit tout le reste. Si nous n'écrivions que des pages statiques, une architecture de présentation nous suffirait. Mais, que ce soit pour traiter un formulaire, retranscrire les données d'une session, initialiser divers paramètres ou orienter un utilisateur suivant son rôle, il va nous falloir manipuler des objets avant même de penser à renvoyer du code HTML : c'est le rôle de l'architecture de traitement.

La distinction entre ces deux types d'architecture permet de bien séparer la logique de la présentation, pour tirer le meilleur parti de l'orientation objet de Zope.

Les deux architectures doivent s'articuler parfaitement pour permettre, par exemple, à une page de traitement d'un formulaire de renvoyer des informations sans qu'il faille générer une nouvelle requête au serveur.

Qualités requises

La pratique a montré que l'architecture sous-jacente d'un site sous Zope doit résoudre les problèmes suivants :

- **Support de pages de traitement et / ou de pages de présentation** : certaines pages effectuent des traitements, certaines présentent des données, d'autres font les deux. Le modèle doit prendre en compte cette caractéristique essentielle d'un site web dynamique.
- **Support de sites existants** : souvent, des pans entiers de sites existent déjà et peuvent être récupérés dans une nouvelle version. Une bonne architecture doit tenir compte de cette contrainte. On notera que ce n'est pas le cas par défaut dans Zope : le plus souvent, les pages importées sont articulées autour de documents `index.html`, or Zope utilise par défaut `index_html`. Nous devrons donc veiller à permettre à Zope de fonctionner comme les autres serveurs web, c'est-à-dire de servir par défaut une page `index.html` dans un dossier.
- **Virtual hosting** : le modèle retenu doit pouvoir s'intégrer dans un environnement de virtual hosting, ce qui signifie que le même serveur Zope doit pouvoir servir des sites web distincts sans difficulté.

> **Remarque**
> *Le virtual hosting désigne la faculté d'un serveur Zope à héberger plusieurs sites distincts sur un même serveur. Pour plus de détails, voir le chapitre 13 consacré à l'administration système de Zope.*

© Éditions Eyrolles, 2001

- **Souplesse** : le modèle doit pouvoir être aisément étendu et modifié. Plus le modèle sera rigide, moins il saura répondre aux besoins ; plus il sera souple et mieux il pourra s'adapter à un grand nombre de projets.
- **Simplicité** : même si l'architecture retenue peut faire appel à un grand nombre de documents et de mécanismes, il faut toujours garder en tête que le site devra être peuplé, maintenu, enrichi *par les utilisateurs*. Il faudra donc veiller à simplifier au maximum la partie qui est vouée à être modifiée par les utilisateurs, les administrateurs, les concepteurs graphiques ou les développeurs chargés de la maintenance.

Nomenclature

Au cours de la réalisation du site, il est important d'élaborer une convention pour nommer les objets : cela facilite la maintenance et permet d'obtenir des informations sur un objet d'après son nom, sans même observer son contenu.

Il est courant que les objets soient préfixés d'un code de quelques lettres permettant d'en déduire la nature. Voici la liste des préfixes couramment utilisés et leur signification :

Tableau 14-2. Préfixes de nommage des objets

Préfixe	Utilisation
`arch_`	*Éléments constituant l'architecture documentaire du site (ce préfixe est surtout utilisé pour des objets* `Folder`*).*
`html_`	`DTML Document`, `DTML Method` *ou* `Script` *(plus rare) renvoyant du code formaté en HTML.*
`proc_`	`DTML Method` *ou* `Python Script` *effectuant un traitement mais ne retournant pas de résultat.*
`nn_`	*(où* `nn` *est un nombre entier à deux chiffres) : éléments devant être ordonnés.*

Toujours par convention, les objets sans préfixe sont les objets qui sont manipulés le plus souvent : ce sont les objets qui représentent le contenu du site.

Enfin, les objets sont en général nommés en minuscules. Les noms de variables (passés au travers d'un formulaire ou créés par `REQUEST.set`) sont au contraire en majuscules.

Racine du site

Afin de voir par le détail comment construire une architecture cohérente, nous allons nous placer dans le contexte du site présenté en étude de cas, `weballage.com`. Il est impératif, lorsque l'on conçoit un site sous Zope, de penser dès le départ au support du *virtual hosting*. Il faut donc créer un dossier spécifique pour chaque site. On peut par exemple créer un dossier `weballage.com` : ce sera la racine du site weballage…

Objet portant une page

Pour éviter les lourdeurs du modèle implicite de Zope – avec l'insertion systématique des appels d'en-têtes et pieds de page standard –, il faut qu'une page soit représentée non pas par

© Éditions Eyrolles, 2001

un DTML Document, mais par un objet conteneur (un Folder fait parfaitement l'affaire). Ainsi est-il possible, avec la méthode index_html, de définir le comportement par défaut d'un dossier puis, au besoin, de surcharger ce comportement. Les URL qui apparaîtront dans le navigateur des utilisateurs pointeront en fait vers des objets Folder, et jamais vers des DTML Document.

Architecture de présentation

Nous venons de voir que l'architecture de présentation comprend toute la partie visible du site, c'est-à-dire, pour ce qui nous concerne, la vue sous forme de code HTML. Lors de la conception d'un site, il est très fréquent de commencer par dessiner une « page témoin », qui servira de référence pour l'intégration de la charte graphique et l'articulation des différents éléments du site ; le site pourra ensuite être aisément modifié de façon qu'il prenne en compte les évolutions de la charte graphique ou de la disposition des pages.

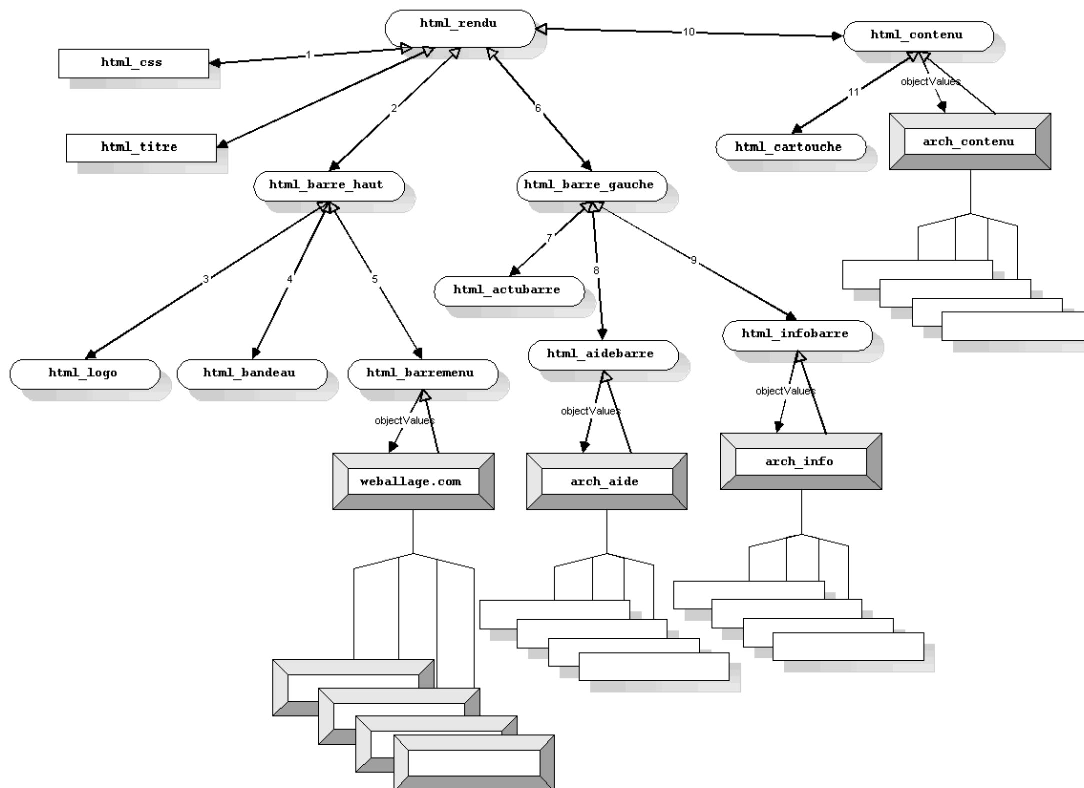

Figure 14-9
Architecture de présentation du site weballage.com

© Éditions Eyrolles, 2001

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.

- Le contenu de la page est le texte qui apporte réellement une information à l'utilisateur.
- Le contenant représente tout le reste, visible ou non : feuille de style, couleurs, découpage en tableaux, cartouche (date et auteur de la dernière modification), barre de navigation…

L'étude de cas du chapitre suivant propose un examen détaillé de chacun de ces éléments. Il est recommandé de dessiner au fur et à mesure de la description de la page la logique associée, c'est-à-dire les étapes nécessaires pour construire une page HTML complète. Voici par exemple le résultat d'une telle architecture pour le site `weballage.com` (voir figure 14-9).

Chaque site, chaque charte graphique, chaque logique de présentation impose ses propres contraintes. Zope ouvre un éventail de possibilités suffisamment large pour laisser au concepteur le choix des solutions les plus efficaces et les plus simples à utiliser.

Architecture logique

Après avoir étudié la partie visuelle du site, il convient d'en étudier la partie logique.

Point d'entrée unique

Que ce soit pour l'architecture de présentation ou pour l'architecture de traitement, il est indispensable d'avoir un point d'entrée unique, qui ne peut être que `index_html` (lors de la conception de la partie visuelle du site, on y place en général du code qui permet d'obtenir simplement le rendu de la page).

Le nom utilisé par Zope pour le document par défaut d'un dossier est `index_html` et pas `index.html`. Cela signifie que si, pour quelque raison que ce soit, un administrateur doit récupérer un site existant depuis un autre outil, utilisant probablement `index.html` comme document par défaut, il ne fonctionnera pas sous Zope. Aussi est-il pratique de ne disposer que d'une seule et unique méthode `index_html`, située à la racine du site, et contenant simplement le code suivant :

```
<dtml-var "_['index.html']">
```

L'inclusion d'un site existant devient alors possible – et la règle est alors la suivante : il ne faut surcharger `index_html` qu'en cas d'absolue nécessité. L'architecture que nous présentons peut fonctionner parfaitement sans que l'on doive définir d'autres méthodes `index_html` ailleurs dans le site.

Méthodes d'initialisation

Lorsqu'un utilisateur saisit un formulaire, il est indispensable de traiter ses réponses avant de rendre quoi que ce soit – ne serait-ce que pour gérer des cas d'erreurs de saisie. Ainsi, il est indispensable de disposer d'une méthode d'initialisation, éventuellement acquise, qui incorpore

© Éditions Eyrolles, 2001

cette logique. Cette méthode ne renvoyant pas de résultat, nous l'appellerons `proc_init`. Ce peut être une méthode DTML ou un script Python, suivant les besoins. Si une page ne requiert pas de traitement particulier mais que `proc_init` est acquis, il suffit de créer une méthode `proc_init` vide dans le dossier correspondant : aucune action ne sera effectuée.

La méthode `proc_init` étant acquise, il suffit qu'un `Folder` la redéfinisse pour que les méthodes `proc_init` de ses parents ne soient pas appelées. Or, il est parfois pratique de disposer d'une méthode qui soit appelée récursivement depuis le début de l'arborescence jusqu'au dossier courant : nous aurons l'occasion d'en rencontrer dans l'étude de cas du chapitre suivant. Il suffit alors d'écrire une boucle qui parcourt les parents de l'objet courant et exécute des méthodes `proc_init_recurse` pour chacun de ces parents. La `DTML Method` appelée `index.html` contient donc le code suivant :

```
<dtml-if proc_init>
  <dtml-var proc_init>
</dtml-if>


<dtml-in "PARENTS" reverse>
  <dtml-if "'proc_init_recurse' in objectIds()">
    <dtml-var proc_init_recurse>
  </dtml-if>
</dtml-in>
```

Les objets `proc_init_recurse` sont exécutés (s'ils existent), à partir de la racine jusqu'au dossier courant.

Interaction traitement / présentation

Il reste enfin à relier cette architecture logique à l'architecture de présentation. Bien qu'en général, un site rende des pages HTML, ce n'est pas là le seul format possible. Par exemple, certaines pages peuvent renvoyer des fichiers, du texte brut… Il ne faut donc pas faire d'assertion quant au format de retour. Lorsque plusieurs choix sont possibles, il faut créer un traitement intermédiaire, ici au travers d'une `DTML Method`, appelée `meth_rendu`, et chargée de rendre la page courante. Dans la plupart des cas cependant, la page retournée est en HTML, et le code de `meth_rendu` est tout simplement le suivant :

```
<dtml-return html_rendu>
```

Si, dans un dossier particulier, il faut renvoyer autre chose qu'une page HTML, il suffit de surcharger `meth_rendu` pour renvoyer un autre type de document.

Enfin, il convient d'ajouter, à la fin de `<index.html>`, la ligne suivante :

```
<dtml-return meth_rendu>
```

La structure résultante est présentée figure 14-10.

© Éditions Eyrolles, 2001

Attention

Nous utilisons ici des balises `<dtml-return>` *et non* `<dtml-var>` *pour éviter de rendre des pages qui contiendraient les sauts de lignes et tabulations que nous avons placés pour indenter le code de* `index.html` *et* `meth_rendu`*. Si ces espaces ne sont pas gênants pour du HTML, ils peuvent le devenir si la page renvoie le contenu d'un fichier binaire (comme c'est le cas pour une page de téléchargement) : les espaces au début du fichier le rendraient illisible.*

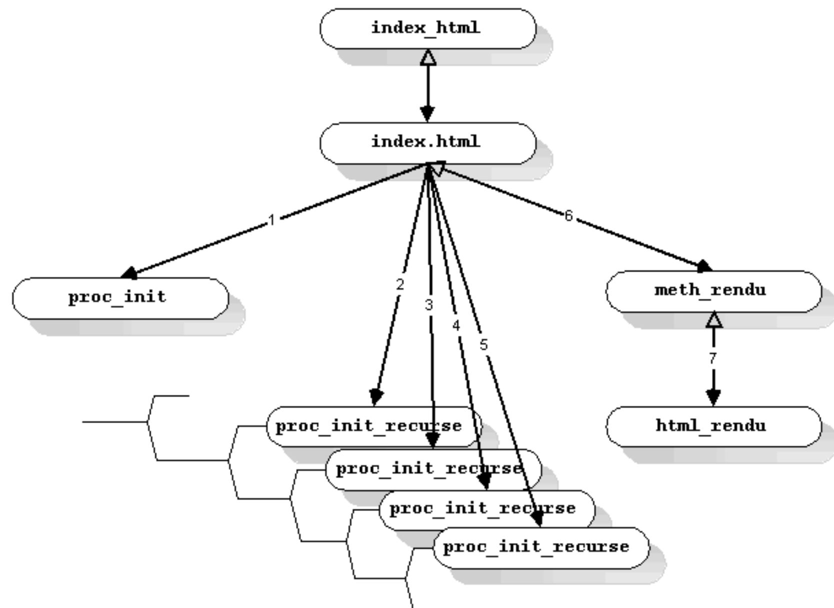

Figure 14-10
Architecture logique du site

Il ne reste plus au concepteur qu'à se pencher sur la partie purement fonctionnelle du site : l'architecture présentée ici est suffisamment efficace pour libérer le concepteur de tâches élémentaires. En outre, le modèle peut être adapté et étendu très facilement.

© Éditions Eyrolles, 2001

En résumé…

Ce chapitre nous a permis d'explorer quelques éléments importants qui doivent être pris en compte lors de la conception d'un site avec Zope.

D'une part, le modèle de sécurité de Zope peut être mieux compris en expliquant les mécanismes liés : acquisition des dossiers `acl_users` et authentification HTTP. L'acquisition permet de garantir des droits en regard du nom d'utilisateur et du mot de passe affectés à un utilisateur. Les mécanismes liés à l'authentification HTTP permettent de mieux comprendre les points relatifs à la sécurité évoqués jusqu'à présent (impossibilité de capturer une erreur d'authentification, impossibilité de proposer une méthode de déconnexion intuitive). La sécurité par programmation peut être assurée avec Zope au moyen des méthodes de `AUTHENTICATED_USER`. En outre, la solution à certains problèmes de sécurité passe parfois par l'utilisation – rigoureusement contrôlée – de méthodes mandataires. L'étude de ces éléments nous a permis de traiter globalement de la sécurité en étudiant les liens entre l'utilisateur authentifié, le propriétaire d'un objet et les permissions affectées aux rôles.

D'autre part, nous avons pu étudier, du point de vue pratique plus que théorique, quelques pistes qu'il s'avère utile de suivre lors de la conception d'un site sous Zope. La nécessité de s'appuyer sur une bonne architecture, extensible, et d'un emploi aisé, conduit souvent les concepteurs à organiser leurs sites selon des critères tirés de l'expérience. La part de la réflexion dans la construction d'une architecture documentaire solide est indispensable si l'on veut qu'en soient garanties les possibilités d'évolution, de maintenance et la facilité d'utilisation. La solution que nous avons présentée ici – et que nous allons mettre en œuvre avec les études de cas – n'est pas *la* solution idéale, mais elle a le mérite de fonctionner et d'avoir passé avec succès les filtres de l'expérience.

Nous avons essayé, tout au long de cet ouvrage, de garder en tête l'aspect pratique de Zope. C'est la raison pour laquelle l'exemple développé au fil de la lecture – un site de vente par correspondance – se voulait réaliste. Les possibilités de Zope sont telles qu'il est tout à fait possible de réaliser sans rencontrer de difficultés des sites de quelque nature que ce soit : portails, commerce électronique, voire applications client-serveur classiques. Aussi, après avoir étudié les aspects techniques de Zope dans les deux premières parties de cet ouvrage, allons-nous maintenant nous consacrer à la pratique, à travers quatre études de cas qui permettront d'approfondir et de consolider les notions étudiées.

© Éditions Eyrolles, 2001