

Plone, un outil de gestion de contenu web

Frédéric Saint-Marcel

INRIA Rhône-Alpes

Frederic.Saintmarcel@inria.fr

Philippe Lecler

IRISA/INRIA

Philippe.Lecler@irisa.fr

Résumé

La gestion de contenu web est devenue une problématique des systèmes d'informations. En effet pour la plupart des organisations, maintenir et faire évoluer un site web demande un effort important.

Les systèmes de gestion de contenu sont arrivés sur le devant de la scène. Ils permettent de séparer le contenu, la logique applicative et la présentation. Ils offrent des interfaces d'administration et de création de contenus qui facilitent la gestion du site. Ils intègrent également un mécanisme de workflow qui permet de définir et d'appliquer une politique de gestion, en spécifiant quelles personnes sont autorisées à modifier certaines parties du site et dans quelles conditions. Enfin ils permettent par la structuration du contenu de rechercher et classer l'information efficacement.

Cet article présente une solution de logiciel libre, le CMS Plone[1][4][5].

Mots clefs

Plone, CMS, CMF, Zope, Archetypes, ArchGenXML

1 Introduction

La première partie de cet article décrira l'architecture logicielle sur laquelle repose Plone. Plone est construit sur le serveur d'applications web Zope (*Z Object Publishing Environment*)[6][7] et la bibliothèque de composants CMF (*Content Management Framework*).

Nous nous intéresserons dans la deuxième partie aux fonctionnalités du site Plone et à sa prise en main. En particulier, nous décrirons sa personnalisation graphique et fonctionnelle.

Enfin, les auteurs feront un retour d'expérience sur la mise en place de Plone dans leurs établissements respectifs.

2 Architecture logicielle

Les différentes briques logicielles constituant un serveur Plone sont décrites par la Figure 1.

Zope est un serveur d'application web, écrit en python pour 90% de son code, les parties critiques en termes de performances étant écrites en C. Il a été développé à

l'origine par *Digital Creations* (*Zope Corporation* depuis 2001) et est passé dans le domaine public, sous licence ZPL (*Zope Public License* : certifiée compatible GPL), en 1998.

Si Zope a été pensé à l'origine pour le développement d'applications de gestion de contenu, le besoin en outils de plus haut niveau s'est rapidement fait sentir. CMF est une boîte à outils ajoutant nombre de services à Zope pour la création d'applications de gestion de contenu.

Plone quand à lui fournit aux gestionnaires de contenu l'interface utilisateur finale prête à l'emploi. Celle-ci est entièrement personnalisable.

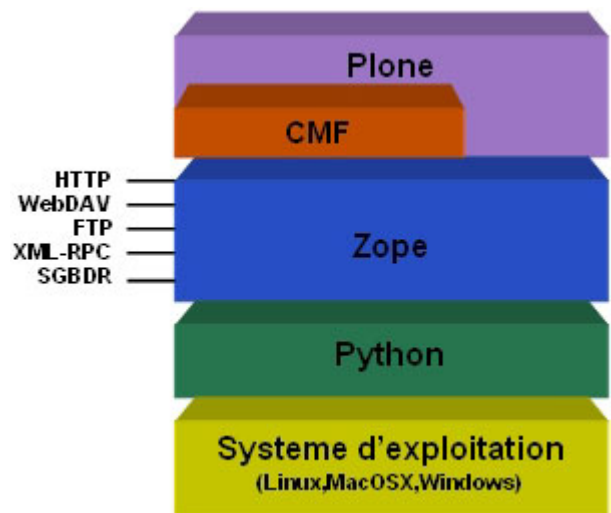


Figure 1 : Les briques logicielles

2.1 Zope : les briques de base

La fonctionnalité première de Zope est la publication d'objets sur le web. Il intègre une base de données orientée objet, historisée et transactionnelle: la ZODB (*Zope Object Database*) contenant non seulement les données de contenu mais aussi des modèles HTML (ZPT : *Zope Page Templates*), des scripts (python, perl), les index du moteur de recherche, des connecteurs à des bases de données externes (SQL).

Zope ne contient pas des pages ou des documents mais des objets rendus persistants grâce à la ZODB. Ces objets sont des instances de classes, le programmeur pouvant définir de nouvelles classes pour créer une application gérant des

objets métier spécifiques à ses besoins. L'ajout, la modification et la destruction d'un objet sont réalisables à travers une interface web : la ZMI (*Zope Management Interface*). Elle est accessible à l'URL http://serveur_zope:port_zope/manage.

Lorsqu'une requête HTTP parvient au ZServer (serveur web de Zope), elle est transformée en appel de méthode sur un objet par le ZPublisher qui est l'ORB (*Object Request Broker*) de Zope. Les objets les plus courants sont ceux de contenu (*Folders, Documents...*) et ceux qui exécutent du code (*Zope Page Templates, scripts python...*). Outre HTTP, Zope supporte également les protocoles FTP, WebDAV et XML-RPC.

La sécurité dans Zope est implémentée à travers les notions de rôles et de permissions. Les utilisateurs (locaux ou enregistrés dans un annuaire) se voient attribuer des rôles auxquels sont associées des permissions. Les rôles locaux permettent de donner des droits spécifiques à un utilisateur dans une partie de l'arborescence du site.

Zope possède un mécanisme d'extension : les *Products*. Dans un *Product*, on définira de nouvelles classes et les modèles HTML (ZPT) permettant l'affichage et la mise à jour des instances de ces classes. Si l'on souhaite étendre les fonctionnalités d'un *Product* existant, on utilisera l'héritage multiple.

CMF et Plone sont des *Products* Zope.

2.2 CMF : composants pour la gestion de contenu

Zope permet de développer des applications web, mais n'offre pas les fonctionnalités de base d'un système de gestion de contenu. C'est là qu'intervient la bibliothèque de composants, CMF, avec les principaux outils suivants :

- L'infrastructure de gestion des utilisateurs (profils) et de leur espace de travail ; la possibilité de personnaliser leur environnement de travail ;
- le *workflow* de publication ;
- l'indexation et le moteur de recherche ;
- la gestion des méta données associées aux contenus conformément au standard *Dublin Core* (<http://dublincore.org/>) ;
- la gestion de l'interface utilisateur ;
- la syndication pour le partage des informations avec des flux XML RSS.

3 Le site web Plone

Plone est entièrement administrable par le web. La gestion du site a deux vues :

- La vue du site web¹ pour la gestion éditoriale (Figure 2) avec les opérations sur le contenu, la configuration des paramètres généraux du portail (gestion des utilisateurs et de leurs droits sur l'arborescence des contenus, options de messagerie, langue par défaut, ...) ;
- La vue technique pour la configuration des composants de Plone via la ZMI.

Nous ne traiterons pas dans le cadre de cet article, la vue du site web. Elle offre toutes les fonctionnalités d'un CMS (*Content Management System*) avec une interface prête à l'emploi.

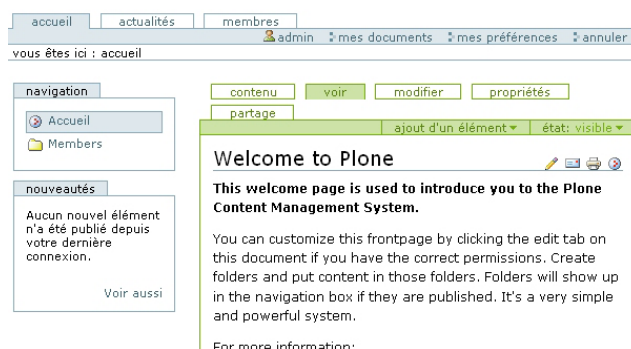


Figure 2 : Gestion éditoriale

La création de l'instance Plone est réalisée au travers de la ZMI (Figure 3). Il suffit de se placer à la racine du site Zope et d'ajouter un objet de type *Plone Site*.

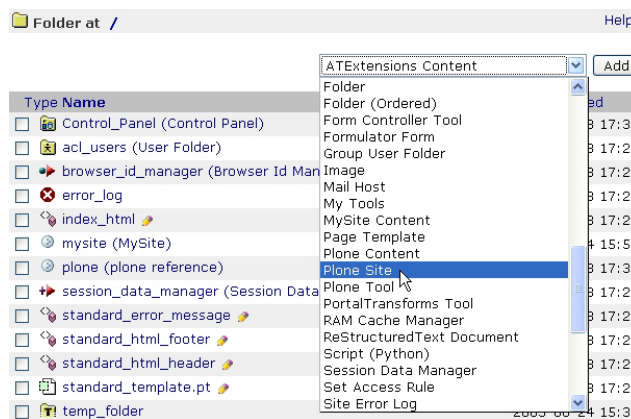


Figure 3 : Ajout d'un site Plone depuis la ZMI

¹ Les CMS possèdent pour la vue du site web un *front-office* pour les visiteurs du site et un *back-office* pour les auteurs qui se sont authentifiés. Plone se démarque des autres CMS en ne proposant pas une charte graphique radicalement différente entre les deux. Le *back-office* se différencie principalement du *front-office* par l'ajout d'onglets (zone verte d'édition des documents) et de raccourcis web pour faciliter la gestion éditoriale.

Un formulaire vous invite à donner l'identifiant de l'objet, le titre donné au site, l'endroit de stockage de la base des utilisateurs et une description sommaire.

Cliquez le bouton *Add* et vous êtes automatiquement dirigé sur la vue du site web qui se trouve à l'URL http://serveur_zope:port_zope/id_objet_Plone_Site/

3.1 Les composants de service dans la ZMI

A la création d'une instance Plone, un certain nombre d'objets sont créés à la racine du site. Ce sont des outils paramétrables prenant en charge les différentes fonctionnalités du serveur. La liste ci-dessous décrit les principaux :

- *portal_catalog* : version évoluée du moteur d'indexation de Zope (*ZCatalog*). C'est à la fois le moteur de recherche pour les utilisateurs et l'outil permettant de créer dynamiquement les pages du site qui sont composées d'éléments vérifiant différents critères (ex: les dix dernières actualités du site, l'affichage d'une partie du contenu d'un document seulement pour les utilisateurs authentifiés) ;
- *portal_types* : interface de gestion des types de contenu ;
- *portal_workflow* : permet de définir le cycle de vie du contenu (ex: un document est tout d'abord rédigé, puis dans un second temps validé et enfin mis en ligne) ;
- *portal_actions* : gestionnaire des actions. Une action permet d'associer de la logique applicative à une interaction web (entre l'utilisateur et le site) ;
- *portal_skins* : gestion de l'interface graphique du site avec le concept de *skins*. Une *skin* est une collection d'objets de présentation (*Zope Page Templates*, feuilles de style) et/ou de logique applicative (*scripts python*) pour respecter la séparation contenu/logique/présentation ;
- *acl_users* : gestion de l'authentification des utilisateurs (locale au portail, annuaire LDAP, base SQL, Active Directory) ;
- *Group User Folder* : permet de créer des groupes d'utilisateurs et de leur associer des rôles (Zope fournit un mécanisme de sécurité qui repose sur trois concepts: les utilisateurs, les rôles et les permissions. Un utilisateur possède un certain nombre de rôles; à chaque rôle est affecté une liste de permissions) ;
- *Portal_syndication* : gestion de la syndication du contenu du site (l'activation se fait uniquement

sur les dossiers). Pour qu'elle soit effective, elle doit être activée de façon globale dans l'outil *portal_syndication* ;

- *Portal_factory* : responsable de la création des objets de contenu.

3.2 Personnalisation

Un des gros atouts de Plone est la possibilité de le personnaliser pour qu'il s'intègre complètement au système d'information de votre institut. Ainsi on pourra modifier la charte graphique, créer des types de contenus en adéquation avec l'information que vous voulez publier et les *workflows* associés qui correspondent à votre propre organisation éditoriale.

3.2.1 Personnalisation graphique

3.2.1.1 Les skins

La personnalisation graphique se fait dans l'outil *portal_skins* accessible depuis la ZMI. On y trouve les principaux répertoires (*sous-skins*) contenant les différents objets graphiques :

- *plone_images* : les images et les icônes de Plone ;
- *plone_templates* : les ZPT définissant l'architecture des pages. Elles sont à Zope ce que sont les JSP au monde Java. Elles permettent de définir des pages dynamiques dont le code est conforme à la norme XHTML ;
- *plone_scripts* : les scripts python utilisés dans les pages ZPT ;
- *plone_styles* : les feuilles de styles de Plone (*plone.css*) ;
- *plone_forms* : les différents formulaires de Plone ;
- *plone_ecmascript* : les scripts javascripts ;

Dans les propriétés de l'outil *portal_skins*, on définit la liste ordonnée des sous-skins (ou *layers*) définies sur le site. Cet ordre servira de règle de recherche des objets graphiques.

On utilise une sous-skin particulière, *Custom*, qui sera en première position de la liste. Lorsqu'on veut modifier un objet graphique, on utilise le bouton *Customize* de cet objet (Figure 4). Ceci a pour effet de le recopier dans le répertoire *Custom* pour le modifier sans altérer le code source de Plone. Ainsi on évitera de perdre ces modifications lors de l'installation d'une nouvelle version.

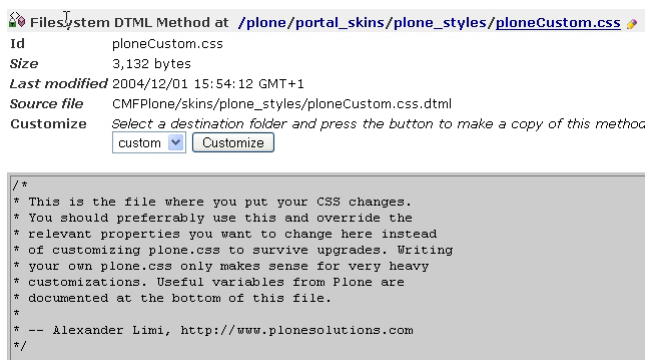


Figure 4 : Customisation de l'objet *PloneCustom.css*

3.2.1.2 Méthodologie pour la personnalisation graphique

La personnalisation peut se faire de plusieurs façons. Les voici, par ordre croissant de difficulté :

- l'objet *base_properties* (sous-skin *plone_styles*) : il permet de gérer avec un formulaire HTML les principaux attributs (police de caractère, couleur, ...) des feuilles de style de Plone (CSS2) ;
- pour les autres attributs CSS : redéfinition/ajout de règles dans la feuille de style *PloneCustom.css* ;
- modification de la structure des pages dans les objets de présentation ZPT : cette technologie se base sur d'autres langages : (1) TAL qui complète le langage HTML en fournissant à ses balises des attributs permettant d'effectuer des actions dynamiques. (2) TALES le langage d'expression utilisé par les attributs. (3) METAL qui permet de réutiliser dans une template donnée le code d'une autre (système de macros).

Extrait de code ZPT :

```
<h1 tal:content="here/title">Titre du document</h1>
```

Lorsque cette expression sera appliquée à un objet ayant pour titre « mon titre », le résultat sera :

```
<h1>mon titre</h1>
```

3.2.1.3 Les gabarits de page

L'infographie classique d'un site Plone est le trois colonnes. Cette structure du site web est mise en oeuvre dans la template *main_template* qui se trouve dans la sous-skin *plone_templates*.

Elle appelle elle-même un certain nombre de templates spécialisées définissant des macros METAL. Exemples : la template *footer* qui contient les informations de *Copyright* et la template *global_searchbox* qui affiche le formulaire de recherche du site.

Les *portlets* Plone ou « boîtes d'informations contextuelles » permettent de modifier le comportement des colonnes droite et gauche afin de les adapter aux différents contextes du site. Les *portlets* sont définis dans la sous-skin *plone_portlets*. La configuration des *portlets*

pour le site est définie dans les propriétés de la racine du site Zope (onglet *Properties*). Les propriétés *left_slots* et *right_slots* permettent de définir l'ordre dans lequel les *portlets* apparaissent (Figure 5). La syntaxe est la suivante : *here/nom_portlet/macros/portlet*.

La création d'une *portlet* consiste en l'écriture d'une page ZPT. On trouvera un exemple de portlet (*portlet_actions*) en annexe.



Figure 5 : Placement des portlets

3.2.1.4 Internationalisation de l'interface

Les libellés de l'interface sont stockés dans des dictionnaires (un par langue) au format *Gettext* que l'on trouvera dans les répertoires *i18n* des différents *Products*. A chaque libellé correspondent deux lignes dans chaque dictionnaire : un id et sa traduction. Par exemple, dans le dictionnaire français on trouvera l'entrée suivante :

```
msgid "My Folder"
msgstr "Mes documents"
```

La Figure 6 montre l'affichage d'un objet de type *portal_tabs* de Plone avec un navigateur web configuré respectivement avec la langue française et anglaise.

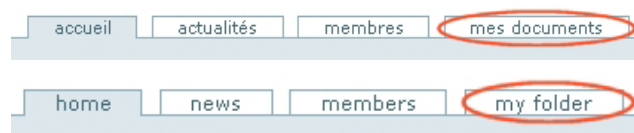


Figure 6 : Internationalisation de l'interface

Lorsqu'on crée ou modifie une template ZPT pour nos besoins on l'internationalisera à l'aide de ces libellés et d'attributs *i18n* dans les balises HTML comme le montre l'exemple suivant :

```
<p i18n:translate="My Folder">sera remplacé par la traduction</p>
```

3.2.2 Personnalisation fonctionnelle

3.2.2.1 Les workflows

Plone propose deux *workflows* : un simplifié pour les conteneurs (*folder_workflow*) et celui par défaut *plone_workflow* pour les autres types de contenu. On associe à un type de contenu un et un seul *workflow*.

Un workflow est un ensemble d'états et de transitions. Un état est une photographie des permissions sur un contenu.

On utilise les rôles auxquels on associe des permissions qui permettent ou non d'effectuer des actions sur le site. Ainsi la consultation d'un document est soumise conjointement aux permissions *Access Contents Information* et *View* et sa modification à la permission *Modify Portal Content*.

Les cinq rôles définis par défaut dans Plone sont les suivants : (1) *Manager*, l'administrateur du site avec « tous les droits » (2) *Reviewer*, les membres du site qui sont chargés de la validation du contenu (3) *Member*, rôle minimal pour contribuer dans le site (4) *Owner*, attribué automatiquement à un membre sur un objet qu'il crée. (5) *Anonymous*, les visiteurs du site (non authentifiés).

La Figure 7 montre les permissions d'un contenu dans l'état *pending* dans *plone_workflow*.

Workflow State at [/plone/portal_workflow/plone_workflow/states/pending](#)

When objects are in this state they will take on the role to permission mappings defined below. Only the permissions managed by this workflow are shown.

Permission	Roles				
Acquire permission settings?	Anonymous	Authenticated Manager	Member	Owner	Reviewer
<input checked="" type="checkbox"/> Access contents information	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> Change portal events	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Modify portal content	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> View	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 7 : Permissions dans l'état *pending*

Dans l'état *pending*, les *Managers* et *Reviewers* ont tous les droits alors que le propriétaire (*Owner*) n'a les droits qu'en lecture.

A un état on associe les transitions possibles. La transition *publish* permet de passer de l'état *pending* à l'état *published* (Figure 8). Elle n'est autorisée qu'aux utilisateurs dont le rôle possède la permission *Review portal content* (*Reviewers*).

Workflow States at [/plone/portal_workflow/plone_workflow/states](#)

pending Waiting for reviewer

- [hide](#) (Member makes content private)
- [publish](#) (Reviewer publishes content)
- [reject](#) (Reviewer rejects submission)
- [retract](#) (Member retracts submission)

Figure 8 : Les transitions possibles depuis l'état *pending*

Créer un workflow revient à définir une machine états – transitions dans l'outil *portal_workflow*, préciser l'état initial, positionner les permissions associées à chaque état et pour chaque transition définir pour quel rôle elles sont autorisées.

La Figure 9 (générée par le *Product DCWorkflow*) représente le graphe d'états/transitions d'un workflow.

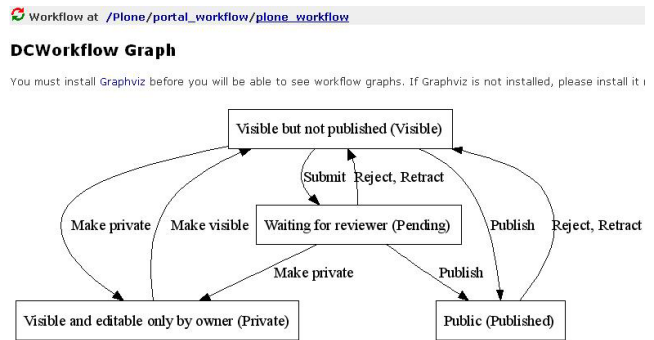


Figure 9 : Graphe d'états du workflow

Il est possible d'associer des scripts python au déclenchement d'une transition qui peuvent par exemple envoyer une notification par mail aux validateurs du document.

3.2.2.2 Les types de contenus

La liste des types de contenus disponibles dans le portail se trouve dans l'outil *portal_types*. Il y a plusieurs méthodes pour créer de nouveaux types, nous présentons ici le framework *Archetypes* [2]. Cette solution définit une API permettant de créer des *Products Zope* et d'automatiser la génération d'une partie du code.

Prenons l'exemple d'un type de contenu nommé *Article*, le programmeur doit créer dans l'arborescence des *Products Zope* un répertoire *Article* structuré comme dans la figure ci-dessous (Figure 10) :

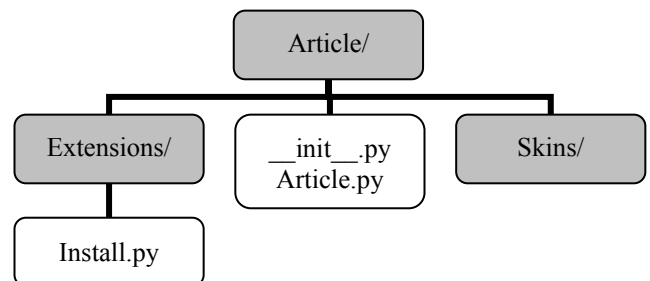


Figure 10 : Arborescence d'un *Product*

- *__init__.py* : le script d'initialisation du *Product* ;
- *Article.py* : la classe du type de contenu ;
- *Skins* : le répertoire pour les modèles ZPT ;
- *Extensions* : le répertoire contenant le script d'installation du *Product* dans Plone.

Le fichier le plus important est *Article.py* qui contient le modèle du document qu'on appelle dans la terminologie *Archetypes* le schéma (Figure 11).

Un schéma est composé de deux notions fondamentales, les champs (*Fields*) et les contrôles (*Widgets*).

Les champs correspondent aux différents types de données contenus dans votre document (ex : un texte, une image, un

fichier attaché) et la manière dont ils sont stockés et gérés par la classe (ex : s'il sont obligatoires ? si il doivent être indexés par le moteur de recherche ?).

Les *Widgets* sont des contrôles HTML utilisés dans le formulaire de saisie de votre document. (ex : une liste déroulante, des radio boutons, une zone de texte multi lignes) Généralement, on part d'un schéma existant que l'on étend. Le schéma minimal est *BaseSchema* qui contient deux champs qu'on retrouve dans tous les types de contenu (*Id* et *Title*).

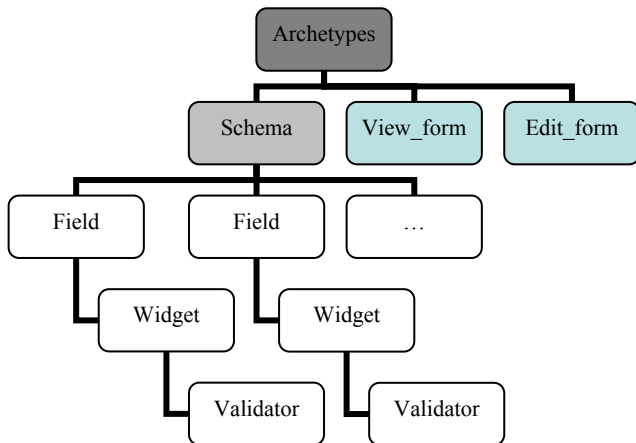


Figure 11 : *Schema Archetypes*

Les composants *Edit_form* et *View_form* correspondent respectivement au formulaire de saisie et à la vue de votre document. Si vous ne redéfinissez pas ces derniers, Archetypes utilisera l'affichage par défaut pour chacun des champs de votre document. On trouvera les principes de redéfinition de la vue (*View_form*) d'un article en annexe.

Dans notre exemple de type de contenu Article, nous définissons cinq champs : ceux de *BaseSchema*, une description, un corps de texte et une date. Le champ description dans le schéma Archetypes est défini comme suit :

```

schema=BaseSchema + Schema((
    TextField('description',
              required=1,
              searchable=1,
              widget =
    TextAreaWidget(
      description="Description de l'article",
      label="Description",
    ),
    .....
  ))
  
```

Le champ description est un *TextField* ce qui correspond à une zone de texte multi lignes, il est obligatoire et son contenu sera indexé par le moteur de recherche.

Le formulaire de saisie par défaut d'un Article est généré automatiquement par Archetypes (Figure 12)

Description ■

Description de l'article

Figure 12 : *Saisie de la description d'un article.*

Dans le schéma on peut aussi utiliser des objets de type *Validator* définissant les contraintes d'intégrité que doivent respecter les données contenues dans les champs. Archetypes propose dix types d'objets prédéfinis pour valider les champs de votre schéma et la possibilité de créer vos propres validateurs. Pour vérifier que le contenu d'un champ est un entier il suffit de placer dans son schéma la ligne :

```
Validators("isInt") ,
```

3.2.2.3 ArchGenXML

Si Archetypes simplifie la définition de nouveaux types de contenu en prenant en charge la majorité du code nécessaire au fonctionnement d'un *Product*, l'écriture du schéma reste délicate du fait de la rigueur syntaxique et de l'absence de messages d'erreur.

ArchGenXML [3] est un utilitaire en ligne de commandes permettant de générer complètement un *Product* à partir d'un fichier XMI (format XML). Ce fichier XMI est l'export d'un modèle UML (Figure 13) décrivant le type de contenu.

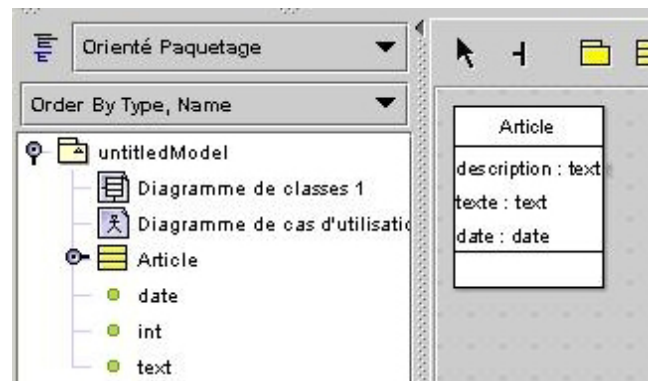


Figure 13 : *Modèle UML de la classe Article*

Chaque champ Archetypes est décrit comme un attribut UML (description, texte, date). Les étiquettes UML définissent les caractéristiques de ces champs (required, searchable...) et les widgets qui leur sont associés (Figure 14).

Marqueur	Valeur
widget:label	"Description"
widget:description	"Description de l'article"
searchable	python:1
required	1

Figure 14 : *Etiquettes UML du champ Description*

Finalement la commande suivante génère le *Product* :

```
ArchGenXML.py -o Article Article.xmi
```

4 Retour d'expérience

Les auteurs de ce papier ont participé à la mise en place du CMS Plone dans leurs établissements respectifs. Le premier est un intranet pour lequel les développements ont été réalisés localement, le second, un extranet dont les développements ont été sous-traités.

4.1 Objectifs

Un des principaux objectifs était de rendre les sites web dynamiques (ex : les actualités publiées dans les différentes rubriques remontent vers la page d'accueil) et vivants avec une mise à jour régulière du contenu.

On devait aussi remédier aux principales difficultés rencontrées par le personnel de nos instituts :

- l'utilisation d'éditeurs pour la création des pages web (Dreamweaver, FrontPage, ...);
- la publication et mise à jour des pages sur le serveur web (problèmes de droits, d'accès, ...);
- l'adoption d'une charte graphique commune.

4.2 Organisation

4.2.1 A l'IRISA

Le gestion du site web institutionnel est inspirée de l'organisation du comité de rédaction d'un journal dans la presse traditionnelle : pour chaque rubrique, on a des auteurs et des responsables de rubrique. Ce sont ces derniers qui décident de la publication des articles et de leur affichage sur la une de la rubrique. Ils peuvent également soumettre les documents leur paraissant importants pour annonce à la une générale.

Les webmasters éditoriaux (service communication) sont responsables de la une générale.

Enfin, les webmasters techniques gèrent l'outil Plone : mise à jour des versions des produits, création - modification des différentes vues...

4.2.2 A l'INRIA Rhône-Alpes

Le site web est l'intranet de l'équipe des moyens informatiques (trois pôles d'activités : système, réseau et support) qui est en charge de la gestion des ressources informatiques. A chaque rubrique on a associé une activité, les auteurs et responsables éditoriaux correspondants. Le webmaster se charge de l'administration de l'outil Plone.

4.3 Le choix de Plone

4.3.1 A l'IRISA

Un groupe de travail constitué de représentants des différents services et équipes de recherche a défini un cahier des charges décrivant les objectifs de l'IRISA. Celui-ci a été envoyé à une vingtaine de prestataires.

Deux solutions ont finalement été retenues : une basée sur Apache Lenya, l'autre sur Plone.

A l'issue de la présentation de ces deux propositions, la solution Plone s'est avérée être la plus mature du fait de sa maîtrise par le prestataire.

4.3.2 A l'INRIA Rhône-Alpes

Deux stagiaires ont été en charge de rédiger le cahier des charges avec l'équipe et de sélectionner quatre CMS (Plone, SPIP, eZ Publish, Typo3) pour monter une plateforme de test. Il devait nous permettre de centraliser plusieurs sources d'informations (pages pour les utilisateurs, compte-rendu de réunions, base de connaissances pour le support) sur un même outil. Le choix s'est porté sur Plone pour sa modularité avec les possibilités de personnalisation, l'ergonomie de l'interface et la pérennité de la solution. La mise en production de l'outil a été effectuée par le webmaster.

4.4 Bilan

4.4.1 Du point de vue auteur

Le contrat est rempli par Plone avec une interface web intuitive et ergonomique qui ne nécessite pas de connaissances technique pour gérer le contenu.

La publication en ligne, la séparation du contenu de la présentation avec les gabarits de pages et les feuilles de style, l'édition des pages avec un éditeur WYSIWYG (Epoz, Kupu) répondent aux besoins cités ci-dessus.

Une phase de formation est nécessaire pour aider les auteurs car il ne faut pas perdre à l'esprit qu'ils devront se familiariser avec un nouvel environnement, des concepts comme les *workflows* et changer progressivement de méthodes de travail.

L'auteur n'est plus seul maître de sa page web, la présentation étant assurée par Plone. Cela a entraîné pour certains une frustration : on leur enlevait un aspect créatif de leur travail.

Les *workflows* introduisent une hiérarchie dans les contributeurs du site. Les responsables des rubriques doivent être réactifs aux soumissions sous peine de goulot d'étranglement dans la publication. Heureusement, il y a remontée d'alerte mail vers les modérateurs lors de la soumission d'un document.

Enfin, reste le problème de la récupération de l'existant. Généralement, celle-ci ne peut pas se faire automatiquement, mais par copier/coller.

4.4.2 Du point de vue administrateur

La mise en production de l'architecture ZOPE/CMF/Plone passe par les étapes traditionnelles de compilation et/ou installation, configuration et sauvegarde des données. Pour des raisons de performance on pourra adjoindre un serveur proxy-cache en frontal (Apache, Squid).

Plone est entièrement personnalisable graphiquement et fonctionnellement. Seul le développement nécessite l'apprentissage de langages (ZPT, python) et la personnalisation nécessite d'être à l'aise dans l'environnement de la ZMI.

A la manière de la création rapide et simple des types de contenus avec l'utilitaire ArchGenXML on peut espérer que les futures versions de Plone minimiseront l'administration à partir de la ZMI.

L'outil permet la gestion des utilisateurs et de leurs droits. Il est possible de se connecter aux différents éléments du système d'information de l'institut (SGBD, annuaire LDAP, ...).

Pour finir, les fonctionnalités de Plone peuvent être facilement enrichies pour proposer un portail web complet. La communauté est très active et propose des produits d'extension variés. (gestion des groupes de travail, wiki, webmail, helpdesk, ...).

Bibliographie

- [1] <http://plone.org>
- [2] <http://plone.org/documentation/archetypes/>
- [3] <http://plone.org/products/archgenxml>
- [4] *Kamon Ayeva, Olivier Dekmyn, Pierre-Julien Grizel, Maik Röder, Zope/Plone.* Les Cahiers du Programmeur Eyrolles, 2004.
- [5] *Andy McKay, The Definitive Guide to Plone.* Apress, 2004.

- [6] *O. Deckmyn, P. Grizel, Zope 2^{ème} édition.* Eyrolles, 2003.
- [7] *Stéphane Bortzmeyer, Zope : un serveur pour applications web.* Dans *Actes du congrès JRES2001*, Lyon, France, décembre 2001.

Annexe

Exemple de création d'une portlet

Définition d'une *portlet* permettant aux membres d'accéder en un « clic » à leur espace personnel et à l'édition d'un contenu de type *Document*.

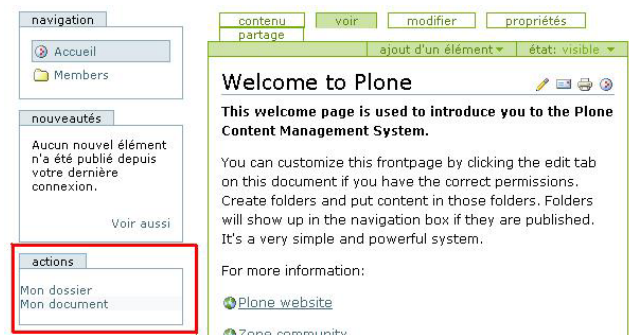


Figure 15 : La portlet actions

1ère étape :

- Créer les actions (*portal_skins>Onglet actions*): « Mon Dossier » et « Mon Document » (Figure 16)
- Faire connaître nos actions dans les actions globales : dans *portal_actions>Onglet Actions providers* rajouter *portal_skins*

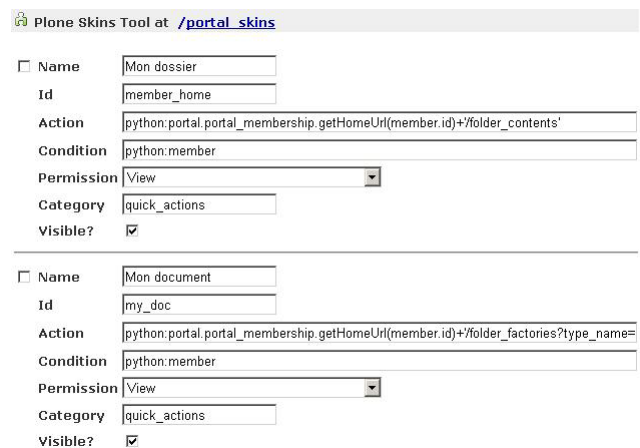


Figure 16 : Ajout des actions

2ème étape :

- Réaliser le *portlet* dans la sous-skin *Custom*



```
Page Template at /portal_skins/custom/portlet_actions
Title
Content-Type text/html
Last Modified 2005-09-09 04:12 PM
Browse HTML source
Expand macros when editing

<html xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      i18n:domain="plone">
<body>
<div metal:define-macro="portlet"
tal:define="actions python:here.portal_actions.listFiltered&actionsFor( here);"
tal:condition="not:here.portal_membership.isAnonymousUser">
<div class="portlet" id="portlet_actions">
<h5 i18n:translate="heading_actions">Actions</h5>
<div class="portletBody"><br>
<div tal:repeat="action actions/quick_actions"
tal:attributes="class python:test(path('repeat/action/even'), 'odd', 'even')">
<a href="#" tal:attributes="href action/url" tal:content="action/name"> liens</a>
<br>
</div><br>
</div>
</div>
</body>
</html>
```

Figure 17 : Le code de la portlet

3ème étape :

- Modifier l'onglet *Propriétés* de la racine de notre site : *left_slots* ou *right_slot* selon la colonne dans laquelle on souhaite voir apparaître le *portlet*.

<input type="checkbox"/> left_slots	here/portlet_navigation/macros/portlet here/portlet_login/macros/portlet here/portlet_recent/macros/portlet here/portlet_related/macros/portlet here/portlet_actions/macros/portlet
<input type="checkbox"/> right_slots	here/portlet_review/macros/portlet here/portlet_news/macros/portlet here/portlet_events/macros/portlet here/portlet_calendar/macros/portlet

Figure 18 : Liste des portlets

Principes pour la personnalisation de la vue d'un Article

La vue par défaut des types de contenu est très simple et il faudra la plupart du temps la personnaliser. La classe *Article.py* doit pour cela importer les classes suivantes

```
from Products.CMFCore.CMFCorePermissions import *
```

et contenir le tuple *actions* suivant qui permet de redéfinir la vue par défaut *base_view* par la vue *article_view*.

```
actions = (
{
'id' : 'view',
'name' : 'View',
'action': 'string:$object_url/article_view',
'permissions' : (View,)
},
)
```

Le fichier *article_view.pt* doit se trouver dans l'arborescence *Skins/Article/* du *Product*. On donne un exemple de la récupération et de l'affichage de la

description et du corps de texte sous la forme d'une table HTML :

```
<table class="listing">
<tr class="even">
<td>Description :</td>
<td><tr tal:content="context/getDescription |
nothing"/></td>
</tr>
<tr class="odd">
<td>Texte :</td>
<td><span tal:content="context/getTexte |
nothing"/>
</td>
</tr>
.....
```

Glossaire

CMS : Content Management System

Zope : Z Object Publishing Environment
(<http://www.zope.org/>)

CMF : Content Management Framework

ZPL : Zope Public License

ZODB : Zope Object Database

ZPT : Zope Page Templates

ZMI : Zope Management Interface

ORB : Object Request Broker

