INJAC : de l'utilisation de Cocoon et J2EE pour la gestion du cycle de vie de documents web.

Brigitte Sor
INPT/ENSEEIHT
Brigitte.Sor@enseeiht.fr

Pierre Gambarotto
Département Informatique de l'ENSEEIHT
Pierre.Gambarotto@enseeiht.fr

Date : 8/10/2003

Résumé

Notre article présente INJAC, application gérant le cycle de vie d'un document pour en faciliter sa publication. En particulier, nous considérons l'existence des acteurs suivants : l'auteur d'un document, le rédacteur en chef, l'éditeur chargé de la publication finale du document, le lecteur.
La finalité de INJAC est d'assister de bout en bout le processus de publication, notamment dans les cas fréquents ou les différents rôles d'auteur, rédacteur en chef, et éditeur sont assurés par la même personne physique.
Toutefois, le modèle implémenté permet si nécessaire de distribuer les rôles de la chaîne de publication à des intervenants différents. Ainsi, pour une publication d'une note de service dans un cadre administratif, l'éditeur sera personnifié par un chef de service vérifiant la validité de la note. Les documents sources peuvent être sous un format quelconque, mais un format facilement convertible en XML (LaTeX, openoffice, rtf) permet d'envisager une conversion automatique ultérieure. Le document final est entièrement en XML.
Ce modèle a été implémenté sous la forme d'un composant JAVA réutilisable. Ce composant est basé sur un modèle de données permettant de représenter des types de documents génériques. Nous utilisons ce composant dans un framework Cocoon qui assure la présentation finale du document, après une éventuelle série de transformations.
Ce service a été conçu pour pouvoir s'intégrer dans des environnements plus généraux de type portail, tel que ESUP-Portail, dans l'optique de la participation à un système d'information global.
Parmi les problèmes traités par cette application, nous abordons plus en détail dans cet article :
– *le stockage des documents,*
– *les liens inter/intra documents,*
– *l'indexation des documents,*
– *le suivi de version et les mises à jours,*
– *le processus éditorial d'un document,*
– *la conversion automatique de format, à des fins de visualisation et/ou d'impression sur des plate-formes matérielles différentes.*

Mots clefs

CMS, Cocoon, XML, Dublin Core, RDF, WebDAV, Workflow, ESUP-portail

1 Introduction

Un site web aujourd'hui doit pouvoir être administré de manière interactive, c'est à dire qu'il doit offrir un "backoffice" permettant de définir de nouvelles pages et rubriques, de mettre à jour et réorganiser les informations. Mais au-delà de ces besoins propres à un site web, les universités comme les entreprises identifient aujourd'hui le besoin de constituer et exploiter un référentiel de contenus, sur lequel s'appuieront des publications de natures diverses : web, édition papier, affichage sur un assistant numérique, restitution audio, etc.
Les produits qui répondent à ces besoins font partie de l'offre produit CMS (Content Management System).
INJAC se positionne dans cette catégorie, avec pour ambition de proposer une mise à jour interactive d'un site web mais aussi la gestion d'un référentiel documentaire.
En effet, INJAC, doit permettre :
– la constitution d' un référentiel de contenus,
– le paramétrage des processus de gestion associés,
– la déclinaison de restitutions de contenus sur une grande variété de médias.
Une autre particularité de INJAC est d'être développé en OpenSource. D'autres produits OpenSource existent mais sont à

notre connaissance centrés sur l'administration interactive de sites web. Citons parmi les plus connus : Typo3[1] ou Spip[2] en technologie PHP.
Le projet INJAC a pris naissance à partir de besoins exprimés par le secrétariat et le corps enseignant du département informatique de l'ENSEEIHT pour couvrir les besoins de l'intranet pédagogique de ce département de formation.
Il a ensuite été intégré dans le projet d'Espace Numérique de Travail ESUP-portail, piloté par l'université de Valenciennes, dans lequel l'INPT est partie prenante.
Cet article est une briève présentation du projet. Nous resituons d'abord les différents termes propres à ce genre d'application. Nous précisons par la suite les principes ayant dirigé l'étude et la conception. La suite détaille l'architecture interne d'INJAC, et nous détaillons pour finir certaines des fonctionnalités du projet. La dernière partie précise quelque peu les différentes technologies mises en œuvre.

2 Terminologie employée

Nous explicitons dans cette section les différents mots-clefs intervenant dans le reste de l'article. Des exemples de mise en oeuvre complètent certaines définitions.

Contexte : Cette notion permet de définir un environnement de travail particulier pour un utilisateur ou un groupe d'utilisateurs. L'objectif de la mise en œuvre de contextes étant de pouvoir présenter à l'utilisateur uniquement les ressources (documents et autres contextes) pertinentes pour lui.

Utilisateur : L'accès aux ressources est protégé par des droits utilisateurs. Ces droits sont étroitement liés aux rôles des utilisateurs dans le système global ou dans un contexte spécifique : créateur, contributeur, éditeur, administrateur. Pour élargir la notion de droits d'utilisateurs, il est aussi géré des groupes d'utilisateurs.

Ontologie : Dans le contexte du web, une ontologie permet de définir les relations entre les différentes parties du vocabulaire et donc de les rendre intelligibles par une machine. Un navigateur d'ontologies permet de naviguer d'un concept à l'autre.

Métadonnées : Une métadonnée est littéralement une donnée sur une donnée. Plus précisément, c'est un ensemble structuré d'informations décrivant une ressource quelconque. Les métadonnées sont en général constituées de mots-clés ou de texte libre. Ces informations peuvent être simples : auteur, date de publication, ou plus complexes et moins aisément définies : annotations par un comité de lecture de tout ou partie d'un document. Les métadonnées sont utilisées dans les systèmes de gestion de contenu (CMS : Content Management Systems) pour éditer, gérer, publier de multiples contenus.

Validation : La validation d'une mise à jour peut impliquer plusieurs intervenants, aux prérogatives distinctes, qui ne seront pas tous disponibles au même instant. Le besoin apparaît souvent de pouvoir séparer les étapes de contribution et de validation. C'est le cas en particulier dans des contextes où la contribution est décentralisée, déléguée à un nombre important d'intervenants. Si l'on souhaite malgré tout garder un contrôle sur les contenus, et c'est souvent fondamental, il faut qu'un document soit d'abord rédigé, puis dans un second temps validé et éventuellement mis en ligne. Cela est d'autant plus important que les contributeurs souvent ne sont pas des spécialistes, ni de la communication, ni du web. Dans certains contextes, ce sont même plusieurs échelons de validation qui sont nécessaires : un contributeur écrit, son chef de service valide à son niveau, et le responsable de la communication valide une seconde fois. La validation doit donc être mise en œuvre par le CMS avec toutes les caractéristiques d'un véritable workflow : les intervenants doivent être avertis par mail des tâches qui leur incombent, et doivent d'un simple clic pouvoir consulter la liste des documents qui attendent leur validation.

Cycle de vie : Au sein de la base des contenus, les documents naissent, vivent et meurent parfois, sous le contrôle du CMS. L'un des besoins les plus classiques consiste à préparer un document avant sa date de publication, et de programmer sa mise en ligne automatique à une date donnée. Un document peut également avoir une durée de vie connue à l'avance : soit en terme de durée, soit selon une date limite. On peut ainsi créer un document intitulé "comment vous inscrire aux JRES2003 ? ", et indiquer dès sa création que ce document est valable 3 mois, ou bien jusqu'au 17 Novembre 2003. Que se passe-t-il alors à cette date ? Le document peut simplement être retiré de la publication de manière automatique. Dans certains cas, le contributeur souhaiterait être prévenu à l'avance de la péremption prochaine. Le CMS pourra donc, dès la création de l'article, permettre de définir sa date de péremption, mais aussi si le contributeur souhaite être prévenu, combien de jours à l'avance, et si le webmaster également doit être prévenu. Une fois averti, le contributeur peut mettre à jour l'article si besoin est, et définir une nouvelle date de péremption. Ce sont là des cycles de vie relativement simples. On peut souhaiter des choses plus sophistiquées, et en particulier qu'entre sa naissance et sa mort, les modalités d'apparition d'un document changent à certaines échéances. Il peut ainsi rester une semaine sur la home page, puis trois mois dans la rubrique actualités, puis un an dans la rubrique archives, et finalement disparaître.

Publication : Le processus de publication peut être décomposé en deux étapes : sélection des contenus et mise en forme des pages. La restitution du contenu (content delivery) consiste à produire des pages sur un média donné, en intégrant le

contenu au sein d'une mise en forme spécifique.

Profils, droits : Les droits de chacun sont définis en référence à l'organisation des contenus. Mais ils doivent également être définis en référence aux actions possibles sur ces contenus. Ainsi tel intervenant pourra proposer un document mais ne pourra pas le valider ou le mettre en ligne. Tel autre pourra valider un article dans les rubriques dont il est en charge, mais ne sera pas autorisé à créer de nouvelles rubriques. Le CMS doit donc offrir une gestion d'habilitation à deux axes : le premier est l'axe des contenus, et de leur organisation, le second l'axe des fonctions liées à la gestion de ces contenus : consulter, mettre à jour, valider, publier ...

Organisation des contenus : Le mode d'organisation le plus classique est bien sûr celui d'une structure hiérarchique arborescente, semblable à la structure de répertoires d'une arborescence fichier. Chaque niveau d'arborescence, chaque division en répertoires, correspond à une logique, un classement par rapport au sens de chaque item, par exemple en référence à des thèmes. Même au sein d'un unique site, l'organisation hiérarchique pure n'est pas toujours satisfaisante, et peut être trop structurante. Il y a des alternatives. On peut par exemple associer un article à différents thèmes, ce qui crée une organisation plus relationnelle (ensembliste) que hiérarchique. Le classement non strictement hiérarchique est souvent une nécessité en restitution. L'article traitant d'un match de foot de l'équipe locale pourra être trouvé à la fois dans la rubrique 'sport' et dans la rubrique 'infos locales'. La possibilité de retrouver un article en différents points de l'arborescence permet de répondre à des logiques diverses de la part des visiteurs. En effet, et c'est un point fondamental, il ne suffit pas de présenter son contenu de manière logique et organisée, il faut aussi que cette logique soit celle du visiteur. Or tous les visiteurs ne raisonnent pas à l'identique, et il faut donc aussi permettre de retrouver un contenu selon différentes logiques.

3 Principes directeurs

La ré-utilisation maximale de standards a été recherchée pour assurer au mieux l'inter-opérabilité de INJAC avec d'autres CMS tant au niveau de la description des données que de leur stockage. Les métadonnées décrivant les documents sont par exemple décrites en XML en utilisant le schéma du *Dublin Core*[3]. Pour l'import et l'export de document, WebDAV[4] peut être utilisé. Ces différentes technologies sont détaillées dans la section 6.
Une autre règle de conception a été de ne pas dupliquer dans INJAC des données existantes dans d'autres lieux du Système d'Information (SI) de l'établissement, pour favoriser le maintien d'une cohérence globale ; citons pour exemple, la réutilisation des données incluses dans le référentiel LDAP, ou dans le module de gestion des groupes d'utilisateurs du portail ESUP. De même, au niveau de l'architecture de l'applicatif, nous avons retenu J2EE et Cocoon qui sont reconnues dans les technologies émergentes dans l'industrie du logiciel pour favoriser une approche globale de type *Modele Driven Architecture* [5].
En ce qui concerne la conception, nous avons adopté la notation UML et nous utilisons l'environnement de développement libre Eclipse[6]. Enfin, un principe méthodologique pour les développements est d'utiliser au maximum les capacités d'abstraction d'un langage objet tel que JAVA. Cela permet sur le long terme de proposer plusieurs implémentation de la même fonctionnalité. Du point de vue de la maniabilité de l'application, il a été recherché une utilisation aisée des principales fonctionnalités même par l'utilisateur occasionnel.

4 Architecture

Nous allons détailler dans cette section les différents choix architecturaux de l'application, obtenus d'une part après étude du cahier des charges, et d'autres part en suivant les principes ci-dessous. Tout d'abord, nous avons favorisé une stricte séparation de la gestion des données et de l'affichage final.
En effet, ce modèle de développement permet de séparer aisément les différentes phases de l'implémentation (modèle de données d'une part, module d'affichage de l'autre). Une conséquence directe est de pouvoir envisager de mettre à disposition plusieurs affichages différents, et de manière générale plusieurs manières de manipuler les données. En particulier, en plus d'un affichage graphique à destination de l'utilisateur final ont été envisagés des outils en ligne de commande, ceci dans le but de réaliser des taches d'administration et d'intégrer l'exécution de ces taches au reste du système, par exemple dans une crontab. Une autre possibilité est d'adapter le module d'affichage en fonction des possibilités du client.
Ensuite, le développement a recours intensivement à des test unitaires, ce qui permet une bonne maintenabilité du code engendré et une localisation des bugs facilité. Pour cela, nous utilisons le framework de tests unitaires jUnit [7].

4.1 Modèle de données

Nous allons maintenant examiner les différentes entités représentées dans le modèle de données de cette application.

Données d'un document. Un document atomique sera physiquement un fichier. L'objet `Données` représente ainsi le moyen d'accéder à un fichier. On dispose ainsi de la possibilité de récupérer le fichier représentant les données. Noter que ce modèle prévoie l'accès au fichier par des protocoles variables. Les méthodes prévues pour cette classe d'objet concernent la récupération du ficher par le protocole spécifié, ainsi que le calcul d'une signature numérique du contenu du fichier.

Document. L'objet de base géré dans notre application représente bien évidemment un document. L'objet `Document` constitue une enveloppe d'un objet `Données`. Par rapport au fichier brut sont ajoutées des notions importantes. Un ensemble de `Metadonnées` vient préciser les propriétés du document, en particulier une référence de l'auteur, un numéro de version, le langage utilisé, le type du document, un titre et une date. L'unicité d'un document sera déterminée uniquement à partir de ces métadonnées, et pas en fonction du contenu des données du document.
Deux modes différents sont prévus pour accéder aux données d'un document. Soit le contenu du document est importé, et donc présent physiquement sur la même machine que les métadonnées, soit le document n'est que lié, et chaque accès au contenu du document passera donc par la récupération des données. Dans le cas d'une importation, les données (en fonction du protocole utilisé) peuvent également donner lieu à une récupération sur une machine distante, mais seul un accès initial est alors nécessaire, les accès futurs correspondant à la consultation d'une copie locale des données.
Ce mécanisme de lien permet d'envisager de lier des ressources externes, et de ne conserver que les métadonnées concernant cette ressource. Le document stocke également une signature numérique associée au contenu des données. Dans le cas de données liées, cette signature permet de détecter tout changement sur le document distant, et de vérifier que les données récupérées lors de chaque accès au document n'ont pas été corrompues, pour cause d'erreur dans le transfert ou par changement du fichier lié. Dans le cas de données importées, cela permet un contrôle sur l'intégrité du fichier, et également de détecter un changement entre la copie locale et l'original du document.

Utilisateur. Cette application doit mettre en relation des documents et des personnes, et logiquement l'entité suivante à représenter est l'`Utilisateur`. Contrairement au document, aucun contenu n'est à récupérer de l'utilisateur (fort heureusement), et l'objet utilisateur ne comporte donc que des métadonnées. L'objet représentant un utilisateur est initialisé par un processus d'authentification. Les métadonnées représentant les propriétés d'un utilisateur (typiquement le nom par exemple) sont donc issues du résultat d'une authentification fructueuse.

Droits d'accès et profils. Dans la mise en oeuvre de la relation document-utilisateur, il faut contrôler les accès possibles de l'utilisateur au document. Pour cela, le premier pas est d'abord de distinguer les différents types d'accès (ou actions) possibles sur un document. Et ensuite d'établir pour un utilisateur ou un groupe d'utilisateur les accès possibles.
Dans cet objectif, on considère la notion de `Profil`, qui relie un ensemble d'utilisateurs et une liste d'action. Le principe est ensuite d'attacher un profil à un document ou à un ensemble de documents, pour ainsi contrôler les accès du groupe d'utilisateur du profil. Il est bien sûr nécessaire d'envisager d'attacher plusieurs profils à un même document ou groupe de documents, pour pouvoir établir un schéma de permission d'accès suffisamment souple, apparenté au système d'ACL (liste de contrôle d'accès) utilisé dans les systèmes de fichiers.

Contexte. La notion de profils fait apparaître la nécessité de regrouper des documents, de toutes façons nécessaire par souci d'organisation des contenus. Emerge ainsi la notion de `Contexte`. De même que les autres entités principales de ce modèle de données, le contexte se voit doté de métadonnées. En particulier, un individu va être distingué comme étant le gestionnaire de ce contexte.
Pour éviter une définition systématique des profils au niveau de chaque document, un ou plusieurs profils peuvent être attachés à un contexte. Un contexte va alors contenir soit des documents avec un ensemble de profils attachés, soit des documents sans profils attachés (ou plus exactement avec un ensemble vide de profils attachés). Si un document ne comporte pas de profils attachés, il va hériter des profils du contexte dans lequel il se trouve. Si un document comporte des profils attachés, soit seuls ces profils sont considérés, soit les profils du contexte sont restreints par ceux du document (ce qui sous-entend bien évidemment une relation au niveau des profils).

Vue d'un système d'information comme un ensemble organisé de contextes. Maintenant que nous avons explicité les relations entre un contexte et ses documents, intéressons nous aux relations inter-contextes.
Deux types de relations vont être considérées entre les contextes. Premièrement, une relation essentiellement technique, qui va hiérarchiser les contextes dans un but organisationnel. De la même manière qu'un document sans profils hérite de ceux du contexte, un contexte va hériter des différentes propriétés du contexte parent. Plus précisément, si aucune métadonnée n'est définie, alors sont reprises les métadonnées du contexte père. On retrouve pour l'héritage de profils les mêmes deux possibilités évoquées pour l'héritage de profil entre un document et le contexte le contenant. Cette relation est appelée relation d'organisation.
Le deuxième type de relation inter-contextes va être utilisé pour exprimer un lien sémantique entre deux contextes. Ceci est purement formel, seule l'utilisation qui sera faite de ce deuxième type de lien apportera réellement un sens. L'objectif de cette deuxième relation est d'introduire la notion de navigation entre les contextes. La navigation inter-contextes doit permettre

à l'utilisateur de sélectionner un contexte, ne lui restera alors qu'à sélectionner un document dans le contexte choisi. Cette relation est appelée relation de navigation.
Le système d'information total est alors constitué d'un ensemble de contextes organisés par la relation de navigation, soit un graphe dont les sommets sont des contextes, et les arcs orientés indiquent la possibilité de naviguer d'un contexte à l'autre. Pour cela, on utilise donc l'objet `SI`, qui en plus de ce graphe comporte également un point d'entrée qui permet de débuter la navigation. A noter que la présence d'une relation d'organisation est imposée dans tous les contextes, sauf dans le point d'entrée qui lui doit contenir une liste de profils. De plus, le recouvrement de l'ensemble des contextes par cette relation doit être tel que tout contexte dispose d'un ensemble de profils, que ce soit en propre ou par exploitation de la relation d'organisation.
Cette architecture très générale peut se simplifier aisément si l'on considère le cas suivant : le SI n'est plus un graphe mais un arbre, dont la racine est le point d'entrée, et la relation d'organisation est déduite de la relation de navigation de la manière suivante : si une relation de navigation existe entre C1 et C2, alors une relation d'organisation existe entre C2 et C1. Dit plus simplement, cela veut dire que l'on va hériter de la configuration du contexte précédemment rencontré pendant la navigation. Ceci est par exemple le modèle implémenté par le framework Slide [8].
Cette simplification permet d'envisager un mapping immédiat entre la structure de données interne et la navigation présentée à l'utilisateur. Notre modèle reste néanmoins ouvert à une implémentation plus générale.

Métadonnées. Pour finir sur l'architecture interne, il reste à expliciter un objet omniprésent dans les différentes descriptions ci-dessus, à savoir les `Métadonnées`. Cet objet est particulièrement simple, puisqu'il est composé simplement du nom de la métadonnée et de sa valeur. Quelques remarques pour dépasser cette vue simpliste : le choix des métadonnées ne doit évidemment pas se faire au hasard. Pour souscrire à un des principes énoncés dans 3, il faut être capable de réutiliser au maximum des ressources externes. Le choix d'un ensemble de métadonnées standard tel que le Dublin Core va clairement dans ce sens, et permettra par exemple de facilement intégrer des documents externes déjà pourvus d'un ensemble de métadonnées. La possibilité symétrique est l'export de données, soit la possibilité pour d'autre CMS d'exploiter les métadonnées. La syndication de CMS est un exemple de cette possibilité. Un autre postulat est la non redéfinition d'une ressource déjà existante. Cela implique directement que la valeur d'une métadonnée pourra être un lien vers une ressource.

4.2 Scénarios d'utilisation typique du modèle de données

Nous allons d'abord décrire un scénario général, avant d'examiner plus précisément les interactions possibles entre les différents éléments du modèle de données dans des scénarios dynamiques.

Exemple général. Examinons tout d'abord un exemple d'utilisation d'Injac. Pour la rédaction d'un document, un contexte de travail est créé par un utilisateur privilégié. Les droits d'exploitation du contexte sont entièrement délégués à deux personnes. Ces deux personnes vont utiliser injac pour échanger les versions successives d'un document. Pendant cette phase de préparation, le document n'est pas publié, i.e. il n'est pas visible par d'autres personnes que les rédacteurs. Injac permet ici de garder les différentes versions du document au fur et à mesure de sa rédaction.
Une fois que les auteurs estiment que le document est prêt, ils vont vouloir le publier. Ils ont ici reçu les plein droits sur le contexte englobant le document. Ils vont donc pouvoir faire évoluer les profils attachés au contexte et au document. Typiquement, ils vont permettre à une certaine catégorie de population (la cible du document) la consultation de la version finale du document. Les droits de consultation peuvent être affinés pour programmer une mise en ligne et un retrait automatiques sur une période donnée. Par exemple, le rédacteur d'un support de TP va le rendre visible sur la période d'enseignement correspondante.
Si les auteurs veulent également publier le document dans un autre contexte, ils doivent proposer le document au mainteneur du contexte visé. Par exemple, le rédacteur du support de TP va vouloir le mettre à disposition dans un espace réservé aux enseignants.

Publication d'un document. Le premier scénario, présenté figure 1, détaille les différentes étapes de la création et la publication d'un document, du point de vue d'un utilisateur.

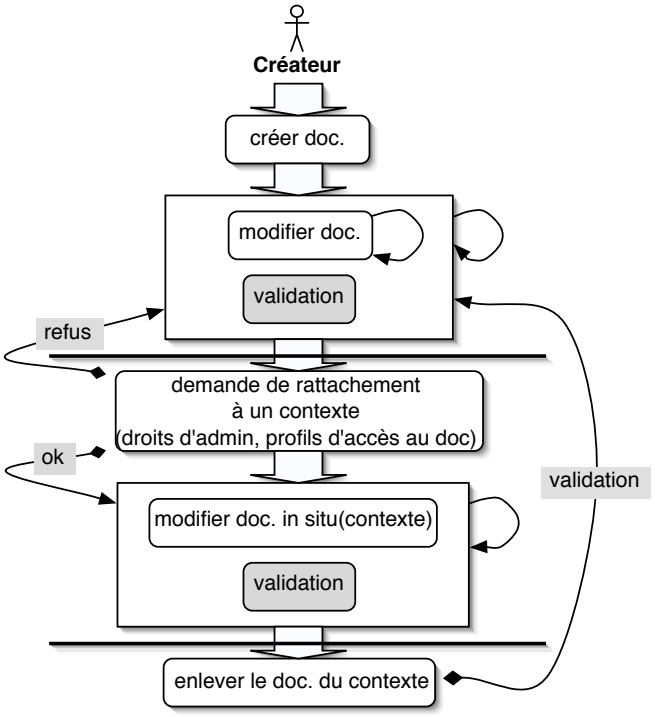

Figure 1 – *Scénario de publication d'un document*

La publication d'un document va mettre en oeuvre plusieurs phases. La première consiste à créer un document, et donc à attacher des métadonnées à un fichier. Le document est alors modifiable à l'infini, et chaque modification est sujette à une étape de validation optionnelle et incrémente le numéro de version du document. La validation correspond à un regard externe pour juger du contenu et de la forme du document, pour des situations où une totale liberté de ton, de forme ou de fond n'est pas de mise. Dans un schéma de publication classique, ce rôle de validation est occupé par un rédacteur en chef par exemple.
La deuxième phase correspond à l'attachement du document à un contexte. Cette phase passe par une demande au référent d'un contexte (trouvé dans les métadonnées du contexte visé). Cette demande précise les droits souhaités sur l'administration du document dans le contexte. Une réponse négative peut être de deux types : soit un rejet total, soit une acceptation avec des droits d'administration différents de ceux demandés. On peut également fixer dans cette demande les profils attachés au document, si l'on souhaite des profils différents de ceux du contexte.
Les possibilités du créateur du document de faire évoluer le document dans le contexte sont entièrement dépendantes des droits d'administration concédés (ou pas) par le responsable du contexte.
Il est possible de renégocier les droits concédés en tentant de nouveau le rattachement du document au même contexte, éventuellement avec des droits différents. Cette possibilité de renégociation peut elle-même être soumise à un droit. Parmi ces droits existe celui de sortir le document du contexte, ce qui met fin à la publication du document.

Manipulation d'un contexte. La création et la publication d'un contexte, représenté figure 2 est similaire à celle d'un document, en plus simple, puisqu'il n'y a pas de contenu à gérer au niveau d'un contexte. La création d'un contexte consiste donc seulement à remplir ses métadonnées.

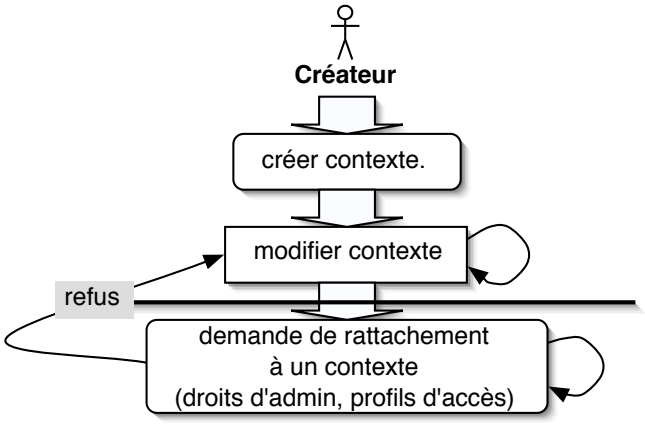

Figure 2 – *Scénario de création et de publication d'un contexte*

Comme pour l'attachement d'un document, les droits du créateur sur l'administration du contexte après son attachement sont à négocier pendant l'attachement. La seule action possible pour changer les propriétés du contexte est donc la renégociation de l'attachement initial. La phase de validation se réduit ici à l'acceptation de la demande d'attachement ou de réattachement. A noter que le processus d'acceptation a pour charge de vérifier qu'un nouveau contexte dispose d'une liste de profils, soit qu'une liste soit fournie dans le contexte, soit qu'un contexte de référence (un est obligatoire) permette de récupérer par exploitation de la relation d'organisation une liste de profils.

Accès à un document. Quel que soit l'accès envisagé à un document[1], le scénario de tentative d'accès à un document ou à un contexte reste le même. Modulo l'existence d'un utilisateur anonyme automatiquement identifié, nous considérons que l'utilisateur demandant un accès s'est authentifié au préalable. Le scénario est représenté figure 3.

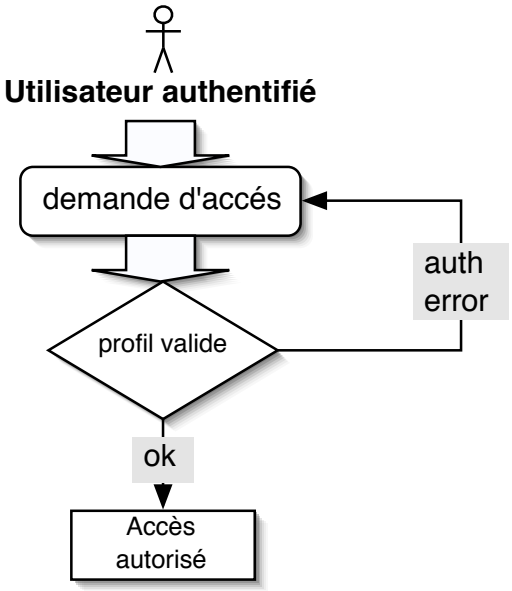

Figure 3 – *Scénario d'accès à un document*

Remarquons que la simplicité du scénario découle d'un certain nombre de propriétés :

– Un contexte ou un document a forcément un ensemble de profils attaché, soit en propre, soit hérité respectivement d'un contexte de référence ou du contexte le contenant.

[1] On parle ici d'un document attaché à un contexte. Pour un document en cours de création, on se réfèrera à la figure 1.

– Plusieurs algorithmes de vérifications des profils sont possibles. Pour l'exemple, les différents critères permettant de distinguer ces algorithmes sont :
  – L'accès est-il permis dès qu'un profil valide est trouvé, ou un nombre plus grand de profils [2]valides est-il requis ?
  – Si un profil valide n'est pas trouvé au niveau courant, doit-on chercher au niveau supérieur ?
  – Si aucun profil valide n'est trouvé, quelle est l'action à faire ?
  – Quelle est la signification de certains profils canoniques, comme les profils avec une liste d'utilisateurs vide ou une liste d'actions vide ?

  Par souci de sécurité et d'efficacité, les critères choisis sont généralement de s'arrêter dès qu'un profil valide est trouvé, de de ne vérifier que les profils au niveau courant. [3] On élimine également les profils canoniques spéciaux, en contrôlant systématiquement à l'attachement des ressources que les profils fournis sont corrects. Notez encore que cette tache n'est à effectuer qu'une fois lors du cycle de vie de la ressource concernée, document ou contexte.

4.3 Rendu graphique

Abordons maintenant la partie rendu graphique de notre application. Les deux éléments du modèle de données pour lesquels une représentation graphique va devoir être produite sont les documents et les contextes.

Rendu des documents. Une des métadonnées caractérisant un document représente le type de ce document. Il est donc envisageable de différencier le rendu du document en fonction de son type.
Le rendu le plus simple, valable quel que soit le type de document, consiste simplement à fournir une interface graphique permettant de récupérer les données associées au document. Parler dans ce cas de rendu graphique est exagéré, puisque les données ne sont pas traitées directement.
Dans la plupart des cas, on va néanmoins vouloir proposer également une visualisation des données du document. Pour cela, deux cas se présentent. Soit le document est dans un format permettant un affichage direct, soit le format n'est pas adéquat et il faut le transformer. Ce deuxième cas est clairement le plus intéressant, surtout si l'on transforme la notion d'*adéquation* en notion d'*adaptation* à l'affichage. On peut ainsi désirer modifier systématiquement les données du document de telle manière que le rendu graphique obtenu soit optimal en fonction des capacités d'affichage de la plateforme. Il est également possible de proposer plusieurs formats d'affichage pour un même document, et soit laisser l'utilisateur décider soit proposer un choix par défaut correspondant aux possibilités d'affichages du périphérique de consultation utilisé.

Rendu des contextes. Avant de pouvoir rendre graphiquement un document, il faut d'abord permettre à l'utilisateur de le trouver.
Deux moyens sont envisageables : le premier est de proposer un accès direct à un document, à partir des résultats d'une recherche sur l'ensemble des documents gérés par l'application, typiquement portant sur les métadonnées du document. Par exemple, on recherche tous les documents d'un même auteur, et on sélectionne ensuite parmi les documents retournés celui que l'on désire afficher.
La deuxième méthode consiste à utiliser la relation de navigation. Le document se trouvant dans un contexte, il faut donc trouver ce contexte, et permettre ensuite de sélectionner le document à l'intérieur de ce contexte. On obtient ici le paradigme de navigation classique : à tout moment on place l'utilisateur dans un contexte, appelé contexte courant, et on lui propose la possibilité de faire évoluer ce contexte courant, ici par l'exploitation des relations de navigations.
Au total, le rendu d'un contexte devra donc comporter :
– un rendu de la liste des documents hébergés par ce contexte, permettant de sélectionner un document et d'effectuer un accès dessus.
– la possibilité d'accéder aux contextes reliés au contexte courant par la relation de navigation.

À noter que cette représentation graphique du graphe de navigation n'a pas à être total. Il est envisageable de ne sélectionner que certaines des relations de navigation, par exemple en fonction des métadonnées des contexts reliés.

5 Fonctionnalités

Rappelons les fonctionnalités principales de l'application telles qu'elles ont été évoquées dans la section 4. La tache principale est de mettre en relation un utilisateur avec un document. Pour cela, l'utilisateur doit être à même de sélectionner le document, et ensuite de le visualiser. Nous allons détailler ici les fonctionnalités annexes.

[2]Eventuellement tous les profils accessibles, soit seulement au niveau local, soit à tous les niveaux exploitables par la relation d'organisation.

[3]La relation d'organisation n'est donc exploité qu'à l'attachement initial d'un contexte au reste, et pas dynamiquement à chaque tentative d'accès. On s'assure de même de recopier dans un document les profils du contexte englobant lors de l'attachement du document au contexte, si le document ne comporte pas de profils.

5.1 Stockage d'un document : version, langue et intégrité.

Tout d'abord, rappelons qu'un document est constitué de métadonnées et de données. Ces dernières constituent un lien vers un fichier qui contient physiquement le contenu du document. Comme expliqué précédemment, ce fichier peut soit être lié, soit importé au sein de l'espace de stockage de l'application. Les métadonnées sont quant à elles de toutes manières stockées localement. Par souci d'efficacité, même un document lié est stocké localement. Plus exactement, un fichier lié n'est pas effacé après la première utilisation (mécanisme de cache).
Sauf à disposer de moyen d'analyse du contenu d'un document, la seule méthode possible pour lier deux documents est l'utilisation des métadonnées.
En particulier, nous voulons pouvoir distinguer deux types de variations sur un même document :
– un changement de version, c'est à dire l'évolution d'un document à cause d'une modification des données associées.
– une version traduite d'un document.
Deux objets document correspondront à deux versions d'un même document si toutes leur métadonnées sont identiques, sauf la métadonnée *version*. Le traitement des langues est plus complexes. Un objet document correspondra à la traduction d'un autre s'il comporte une métadonnée faisant référence à l'autre document.
Ces deux types de variations peuvent entraîner la présence au sein d'un même contexte de plusieurs versions d'un même document. Le modèle de données et les métadonnées décrites ci-dessus permettent de différencier ces différentes versions mais également de les relier entre elles. Le rendu d'un contexte pourra donc utiliser ces informations pour refléter à l'utilisateur les liens détectés entre ces différentes versions.
Au niveau stockage, la gestion des versions sera différente selon que les données sont liées ou importées. Si les données sont importées, on pourra stocker les différentes versions. Si les données sont liées, un changement de version correspondra à un changement sur le fichier lié. On pourra donc détecter ce changement, grâce à la signature du fichier, mais pas conserver les versions précédentes.
Au total, par l'utilisation de certaines métadonnées, on offre un système gérant à la fois l'évolution d'un document et ses possibles traductions. Au niveau de l'utilisation finale, le rendu graphique d'un contexte incorporant des versions multiples d'un même document fera apparaître un seul document, et permettra de choisir la version souhaitée.

5.2 Séparation du fond et de la forme

Le contenu d'un document et sa présentation sont deux choses différentes, et relèvent de métiers distincts. Ils doivent pouvoir évoluer différemment. Il est souhaitable de pouvoir changer le design d'un site sans avoir à intervenir sur les contenus et réciproquement. Une mise en page unique est fréquemment partagée par de multiples contenus, cela permet d'assurer une homogénéité graphique d'un site. La mise en forme se définit en référence à un média de publication et l'on ne veut pas réduire la possibilité de publication à un seul média.

5.3 Structuration des contenus

La description de la structuration sera faite sous forme fonctionnelle suivant le schéma Dublin Core.
En environnement J2EE, ces types spécifiques seront définis comme des objets Java.

5.4 Organisation des contenus et organisation des publications

Les contenus seront organisés suivant une structure hiérarchique arborescente en utilisant les schémas RDF. Il n'y a pas de correspondance figée (mapping) entre organisation des contenus et organisation de la publication pour que les mêmes contenus puissent suivre plusieurs publications. Cette fonctionnalité repose sur l'utilisation du framework Cocoon.

5.5 Gestion des éditeurs ou contributeurs

Les droits de chacun sont définis en référence à l'organisation des contenus, et aux actions possibles sur ces contenus. Ceci suppose une gestion d'habilitation à deux axes : celui des contenus et de leur organisation et celui des fonctions liées à la gestion des contenus.

5.6 workflow de validation

Les étapes de contribution et de validation sont séparées pour la prise en compte de contributions décentralisées, déléguées à un nombre important de contributeurs internautes. Les intervenants dans le schéma de validation sont avertis par mail des tâches qui leur incombent et peuvent consulter aisément la liste des documents en attente de validation.

5.7 cycle de vie des documents

Pour ne pas restreindre les utilisations possibles de ce projet, les étapes suivantes doivent être possible dans le cycle de vie d'un document :
– il est possible de préparer un document avant sa date de publication et de décider de sa mise en ligne à une date donnée

– un document peut avoir une durée de vie connue à l'avance ; soit en terme de durée, soit selon une date limite
– un document peut apparaître dans des contextes (rubriques) différentes : Par exemple, une semaine sur la page d'accueil puis un mois à la rubrique actualités puis un an à la rubrique archives pour disparaître enfin.

5.8 Publication

Sélection des contenus. La base des contenus est alimentée au fil de l'eau et les contenus sont positionnés avant publication dans les différents contextes d'édition. Les classifications transverses qui ne dépendent pas de l'organisation principale du site sont prises en compte. On devra par exemple pouvoir sélectionner les documents rédigés par tel contributeur et qui ont moins de 6 mois.

Restitutions et gabarits. Pour une page donnée, un ou plusieurs gabarits seront utilisés définissant le positionnement et les attributs de mise en forme. Ceci est réalisé Sous Cocoon par l'utilisation de modèles de styles au format XSL. Les transformations successives d'un document permises par Cocoon permettent également de supporter plusieurs média de publication, et de ne pas se limiter aux seules pages web. Par exemple, XSL/FOP permet de transformer un document XML en pdf.

Personnalisation de la restitution - Intégration à ESUP-portail. L'intégration de INJAC dans ESUP-portail permet de récupérer le profil de l'utilisateur et ainsi de différencier la restitution suivant ce qu'il souhaite voir et est autorisé à voir, à partir d'une base de contenus unique administrée de manière globale.

5.9 Echanges de contenus et syndication

Il est courant qu'un site ait à échanger des contenus avec d'autres sites (mise à disposition ou reprise). Ceci devrait être de plus en plus fréquent dans notre communauté avec entre autre le déploiement d'universités numériques en région. Un moyen normalisé d'échange doit être utilisé à cette fin sous format XML (DublinCore, RDF). Cette indexation partagée permettra de disposer d'un outil de recherche puissant et fiable.

6 Choix technologiques

6.1 Cocoon

Le projet Apache *Cocoon*[9] est un système de publication XML sous licence Apache pouvant interagir avec de nombreuses sources de données dont les systèmes de fichier, les bases de données relationnelles, les annuaires LDAP, les bases de données XML et des sources de données réseau et générer des contenus sous de nombreux formats dont HTML, WML, PDF, SVG, et RTF.
Cocoon a pour but de séparer le contenu, les styles, le code et la gestion des fonctions dans un environnement XML. Il est aussi bien adapté pour des sites statiques que pour des applications dynamiques.

Principe de Cocoon : Une application Cocoon est constitué de plusieurs pipelines (chaîne de transformation). La requête envoyée par le client http est interprétée par des *Matcher* qui aiguillent les requêtes vers les pipelines adéquats. Le pipeline sélectionné va alors effectuer plusieurs opérations. Tout d'abord, il va générer un flux XML à partir d'un fichier statique (XML ou autre), d'un fichier de code java (jsp ou xsp), d'une base de données XML ou plus généralement à partir d'une classe Java. Cocoon offre un grand nombre de *Genrator*, et il est relativement aisé d'en développer un pour ses propres besoins. Ensuite, le flux XML va être transformé en un autre flux XML grâce à des *Transformer*. L'effet d'un *Transformer* est fréquemment l'application d'une feuille de style XSL (XSL stylesheet). Il est possible de regrouper plusieurs flux, en provenance directe de *Generator* ou à la sortie d'un *Transformer*. Enfin, le flux XML va être sérialisé grâce à un *Serializer* pour prendre le format attendu par le client. On peut résumer ce principe de fonctionnement par le schéma de la figure 4.

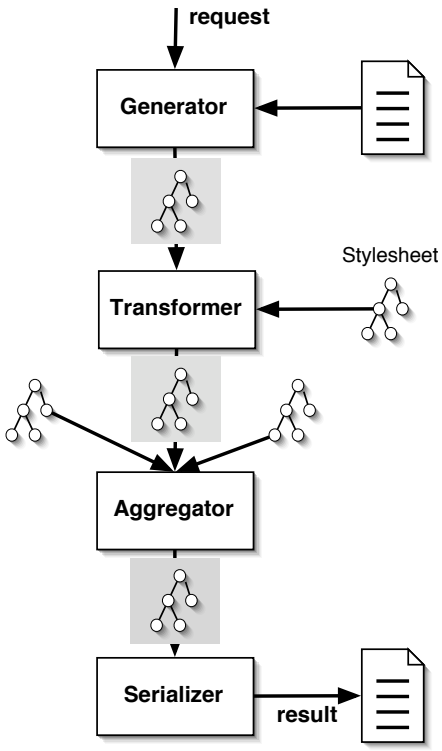

Figure 4 – *Pipeline Cocoon*

Mise en oeuvre de Cocoon : Toute la configuration de Cocoon se fait dans un fichier XML. Il est facile de modifier par ce fichier la configuration d'une application Cocoon. Il contient la déclaration de tous les composants cités précédemment (*Matcher*, *Generator*, *Transformers*, *Serializers*, mais aussi des *Readers* et des *Selectors*). On va aussi y trouver la liste de tous les pipelines qui contiennent les patterns d'URL à traiter.
Au niveau de la programmation, Cocoon propose un langage de remplacement de JSP : XSP. On retrouve en fait les mêmes possibilités qu'avec JSP mais les commandes sont intégrées dans un espace de nom XML particulier. Les fichiers XSP dont lus par des Generators pour créer un document XML. Comme pour les JSP, les fichiers XSP peuvent contenir beaucoup de code Java mélangé à d'autres information en XML. Pour garantir une bonne séparation entre la logique et le contenu de l'application Web, il est néanmoins recommandé de minimiser le code Java dans les XSP. Mis à part une meilleure intégration dans Cocoon, un des gros avantages de XSP sur JSP est la possibilité d'utiliser des feuilles de style XSL. Les feuilles de style sont des fichiers de transformation XSL que l'on va appliquer aux fichiers XSP. Ainsi, on va pouvoir factoriser le code produit et gagner du temps lors de l'écriture des fichiers XSP.
En outre, Cocoon contient un grand nombre d'outils qui peuvent aider à la réalisation d'applications Web. Ainsi, Cocoon propose un outil de gestion de Sessions très complet. On peut aussi utiliser les outils d'authentification fournis. Pour gérer les formulaire, Cocoon propose JXForms, un framework qui se base sur Apache JXPath et W3C Xform.
En ce qui concerne le domaine des CMS, Coccon présente également l'intéret de faciliter l'uniformisation de l'interface et sa modification. En effet, l'idée de Cocoon est de travailler avec des documents XML non mis en forme et d'appliquer la mise en forme au dernier moment, lors de transformations XSLT. Ainsi, on va retrouver les informations de mise en forme dans seulement quelques fichiers XSL. On est donc sûr que toutes les pages de l'application Web vont respecter la même charte graphique. Et lorsque l'on voudra changer cette charte graphique, il n'y aura que peu de fichiers à modifier. Cette possibilité est d'autant plus intéressante que le CMS est destiné à des universités ou services qui ne souhaiterons pas nécessairement la même charte graphique.

6.2 RDF/Dublin Core

Pour sélectionner des documents et pallier les limitations des moteurs de recherche qui travaillent en "indexation plein texte en aveugle", une voie consiste à structurer le web pour rendre explicites les relations sémantiques qui existent entre les unités documentaires qu'il contient. RDF (Ressource Description Framework)[10] s'inscrit dans cette approche.
Cette approche passe par la définition de schémas structurant l'information disponible et le choix d'indexer certaines unités

d'information pour permettre la recherche des entités les contenant. La communauté des documentalistes et bibliothécaires, depuis longtemps sensibilisés à cette problématique, ont proposé en 1997 un schéma descriptif normalisé ayant vocation à pouvoir décrire toutes les ressources documentaires du web : le Dublin Core [11].
Le Dublin Core propose quinze propriétés descriptives, relatives soit au contenu, soit à la propriété intellectuelle, soit aux caractéristiques physiques de la ressource décrite. Le Dublin Core peut être exploité en utilisant les éléments META de html. Sa définition est purement sémantique. Aucune hypothèse n'est faite sur les outils logiciels et les langages utilisés.
RDF offre quand à lui, une méthode générale pour représenter et exploiter un tel schéma ou d'autres plus complexes. En effet, un schéma descriptif tel que le Dublin Core est indispensable mais non suffisant. Dans INJAC, il nous faut aussi pouvoir définir des schémas plus spécifiques, correspondant aux besoins par exemple d'une catégorie d'utilisateurs. Tout système de structuration d'informations doit être fondé sur un système formel aux propriétés mathématiques bien définies. RDF est basé sur un modèle de graphe étiqueté orienté. La syntaxe XML est une transcription directe du modèle formel décrivant le graphe des propriétés sous forme de triplets (prédicat, sujet, objet).

Schémas RDFS Les déclarations RDF interviennent pour la description des métadonnées des contextes (nœuds du graphe) et des documents (feuilles du graphe). Les déclarations RDF définissent des relations entre des objets qui appartiennent à un univers sémantique particulier ; par exemple celui des cours, TP, TD sur le thème de la "programmation objet". Un tel domaine, pour lequel sont définies des propriétés spécifiques et des catégories conceptuelles est appelé un schéma RDF (RDFS). RDF et RDFS offrent des primitives pour définir les schémas répondant à nos besoins applicatifs. Ces primitives sont de trois types :

– celles qui définissent les ensembles qui fondent le modèle de représentation des connaissances,
– celles qui définissent les relations générales entre objets de ce modèle,
– celles qui permettent de formuler des contraintes sur certaines propriétés ou sur certains objets définis par spécialisation.

RDF est surtout utile pour partager des données ; car qui dit partage, dit langage commun. Le logiciel le plus familier sans doute aux utilisateurs de notre communauté utilisant RDF est RPMfind. Ce logiciel permet aux développeurs et aux utilisateurs du monde Unix de localiser des packages au format RPM disponibles sur des serveurs distants. Les propriétés descriptives de ces packages sont décrites en RDF (disponible dans les distributions Linux utilisant le format RPM).

6.3 WebDAV

Webdav est un protocole applicatif défini comme une extension de http et utilisant XML pour le corps des messages. Il utilise un système de métadonnées nécessaire à l'édition coopérative et à la gestion de versions de documents sur le web.

Fonctions de base. Les fonctions basiques assurées par le protocole WebDAV sont :
– création et accès à des métadonnées d'identification de ressources ou collection de ressources ; WebDAV propose un formalisme général de représentation et un accès aux métadonnées attachées aux ressources décrites par des liens typés.
– copie ou déplacement d'une ressource sans rendre périmées les méta données qui la décrivent.
– obtention selon une méthode standard de la liste des ressources contenues dans un répertoire en ligne.
– mécanisme de préemption (verrouillage exclusif ou partagé) lors de l'édition d'une ressource par un ou des utilisateurs autorisés.
Les trois premières fonctions permettent de considérer un répertoire partagé accédé par http de la même façon qu'un répertoire du système de fichiers local.

Principes. Les ressources de tout type déposées sur un serveur http vont être décrites à l'aide de propriétés choisies par l'administrateur du site. Des propriétés peuvent également être associées à une collection de ressources. Un certain nombre de propriétés sont prédéfinies par la norme pour permettre de gérer les droits d'accès en lecture et écriture aux ressources ainsi que leur verrouillage.
Ainsi, une application WebDAV va pouvoir :
– définir des propriétés,
– affecter des valeurs à celles-ci,
– consulter des valeurs de propriétés,
– créer des collections,
– déposer des ressources sur le serveur, les déplacer, les effacer,
– verrouiller ou déverrouiller des ressources et des collections.
Ces opérations sont obtenues à l'aide de méthodes http. WebDAV rajoute des méthodes formées d'un entête et d'un corps, ce dernier étant une ressource XML bien formée. Les réponses à une méthode sont également des ressources XML bien formées.

7 Conclusion

A partir d'un besoin local, le projet INJAC s'est fixé comme objectif de proposer une plateforme de publication basée sur des standards ouverts. Les objets sont manipulés par l'intermédiaire de métadonnées facilement échangeables (XML et RDF). Les choix technologiques (J2EE et Cocoon) permettent une évolution progressive et ouverte des fonctionnalités. Ce développement s'inscrit dans un contexte complètement ouvert et toutes les contributions sont les bienvenues[4].
De manière plus prospective, notre travail peut être vu comme une transformation de modèles. Dans cette optique, les technologies émergentes des MDA (Modele Driven Architecture) peuvent constituer des éléments permettant d'associer une méthodologie à ces transformations.

Références

[1] Mickael Kasper. Typo3. page Web. http ://www.typo3.com.

[2] Spip Group. Spip. page Web. http ://www.spip.net.

[3] DC Initiative. Dublin core metadata initiative. page Web. http ://dublincore.org.

[4] WEBDAV. Web-based Distributed and Versionning. page Web. http ://www.webdav.org.

[5] OMG. MDA Model Driven Architecture. page Web. http ://www.omg.org/mda.

[6] IBM. Eclipse. page Web. http ://eclipse.org.

[7] Erich Gamma, Kent Bech. junit. page Web. http ://junit.org.

[8] Jakarta Project. Slide. page Web. http ://jakarta.apache.org/slide/.

[9] Apache Project. Cocoon. page Web. http ://cocoon.apache.org.

[10] Ralph R. Swick Ora Lassila. Ressource description framework. page Web W3C Recommendation, February 1999. http ://www.w3.org/TR/REC-rdf-syntax.

[11] Alain Michard. XML langage et applications. Eyrolles, 2001.

[4] Nous tenons déjà à remercier les relecteurs du présent article, notamment Philippe Mauran, Gérard Padiou et Philippe Queinnec.