Presses
universitaires
de Caen
Maison de la Recherche en
*Sciences Humaines (*mrsh*)*
14032 Caen cedex

Davy
DERMILLY
Master MI3
Année 05/06

Rapport de stage

Élaboration d'une application dédiée à la dtd onix

Maître de stage : Pierre-Yves Buard

Enseignant tuteur : Jacques Madelaine

Presses
universitaires
de Caen
Maison de la Recherche en
*Sciences Humaines (*mrsh*)*
14032 Caen cedex

Davy
DERMILLY
Master MI3
Année 05/06

Résumé :

Les étapes de fabrication d'un livre sont nombreuses et le nombre de données qu'elles génèrent nécessite un système performant pour les mettre à jour. De plus, le développement des réseaux offre maintenant de telles possibiltés que les besoins en échanges d'informations ont fortement évolué. Ces échanges peuvent être faits de différentes façons. Une solution utilisant la norme internationnale onix a été retenue lors d'un précédent projet. Ce rapport propose de reprendre l'analyse de celui-ci en la confrontant aux besoins réels des utilisateurs des Presses universitaires de Caen telle que la création d'un système centralisé qu'ils pourront mettre à jour.

Mots clés :

Java, j2ee, jaxp, jsp, xml, onix, eXist, Struts, Axis, mvc, php, mySql

Remerciements

Je tiens à remercier mon maître de stage, Pierres-Yves, pour sa grandes disponibilité.

Table des matières

Liste des illustrations

Introduction

L'objectif de ce stage est de reprendre et finaliser mon projet individuel. Celui-ci avait pour but la mise en œuvre une application Web qui permet à tous les acteurs de la chaîne de production d'un livre d'accéder aux informations éditoriales le concernant.

Ces informations peuvent être son titre, son auteur, son numéro ISBN, sa mise en page, le type de reliure, son type (revue, roman...), son prix, sa disponibilité sur le marché, sa couverture et bien d'autres.

Les acteurs de cette chaîne de production sont nombreux. Le livre part d'un auteur ou d'un groupe de recherche, pourra passer entre les mains d'illustrateurs, éventuellement d'un traducteur, Il sera relu et corrigé, mis en page puis imprimé. Pour finir un diffuseur se chargera de la promotion et un libraire de le vendre. Et il existe encore bien d'autres intervenants.

L'échange et l'organisation de toutes ces informations seront faits par l'intermédiaire de fichiers XML suivant le standard ONIX[1].

L'idée principale étant de placer ONIX au cœur du système.

[1] ONline Informations eXchange.

Chapitre 1

Présentation du sujet

1.1 Contexte

1.1.1 Les Presses universitaires de Caen

Les Presses universitaires de Caen sont un service commun de l'Université de Caen. Elles éditent et diffusent les travaux des équipes de recherche de l'Université. Ce service de dix personnes est réparti de la manière suivante :

- une cellule *édition* qui assure le travail éditorial de la relecture du manuscrit jusqu'au dépôt légal ;
- une cellule *gestion commerciale* chargée de la vente des ouvrages soit directement, soit en librairie ;
- un secrétariat ;
- la comptabilité.

Tout projet déposé aux Presses universitaires de Caen doit au préalable avoir obtenu la caution scientifique d'une composante de l'Université[1] qui sera chargée d'en assurer le financement.

Le texte à éditer est alors soumis à l'approbation d'un comité de lecture, nommé par le Conseil scientifique de l'Université, qui se réunit plusieurs fois par an.

Chaque année, les Presses universitaires de Caen éditent entre vingt et trente ouvrages. Les Presses universitaires de Caen, en leur qualité d'éditeur, doivent gérer toutes les informations qui se rattachent à l'édition d'un livre. Il s'agit, non seulement d'informations propres à la structure du livre (titre, auteur, résumés...), mais aussi d'informations données par tous ceux qui sont intervenus dans sa fabrication (référence, prix et quantité en stock, donnés par le distributeur, numéro ISBN[2] fourni par la BNF[3]...).

[1] Une équipe de recherche la plupart du temps.
[2] International Standard Book Number.
[3] Bibliothèque Nationale de France.

1.1.2 Les différents types de production

Les documents édités par les Presses universitaires de Caen peuvent être classés en trois catégories :

- les monographies, livres écrits par un seul auteur ;
- les collectifs, livres écrits par différents auteurs sur un thème commun ;
- les numéros de revue, composés d'articles écrits par différents auteurs.

À l'intérieur de chaque catégorie, les documents peuvent avoir des formes diverses : reliés ou non, un ou plusieurs volumes, avec ou sans CD-rom, support papier ou numérique...

Quelle que soit la forme finale, un projet est toujours composé d'un texte à relire et à mettre en page ainsi que d'une couverture (sauf pour les editions numériques) ou d'une page de présentation.

Par commodité, tous les types de documents édités seront appelés *ouvrages*, quel que soit le support de publication.

1.1.3 Les différentes étapes de la chaîne éditoriale

Dépôt du manuscrit

L'*auteur* écrit le texte. S'il s'agit d'une œuvre collaborative, il peut y avoir plusieurs auteurs, avec, le cas échéant, des fonctions distinctes : *directeur scientifique*, *rédacteur*, *illustrateur*, *traducteur*...

L'équipe de recherche amène son projet éditorial puis paie les frais d'édition et de fabrication. Un comité de lecture est chargé de valider la scientificité des ouvrages et leur compatibilité avec le catalogue des Presses universitaires de Caen.

Production

Si l'ouvrage reçoit un avis favorable, différents acteurs interviennent :

- un correcteur relit le texte et corrige les fautes ;
- un opérateur PAO, en collaboration avec un graphiste, assure la mise en page et la conception éventuelle de maquette de couverture ;
- un chargé de fabrication coordonne l'ensemble.

1.1.4 Commercialisation

Une fois le livre réalisé, il est mis en vente par l'intermédiaire du réseau diffuseur-distributeur-librairie. Parallèlement, sa promotion est assurée auprès des divers médias concernés.

Le diffuseur : il est chargé, par l'intermédiaire de ses représentants, de la promotion du livre. Il organise des campagnes promotionnelles, assure de la mise en place du livre dans les différents points de vente et contrôle le réassort.

Le distributeur : il a un rôle logistique. Il gère le stock de livres pour le compte de l'*éditeur*. C'est lui qui reçoit et expédie les commandes et se charge de la facturation.

Le libraire : il vend directement au public, mais aussi aux bibliothèques qui offrent un accès non marchand au livre.

1.1.5 Schéma récapitulatif de la chaîne de fabrication d'un livre

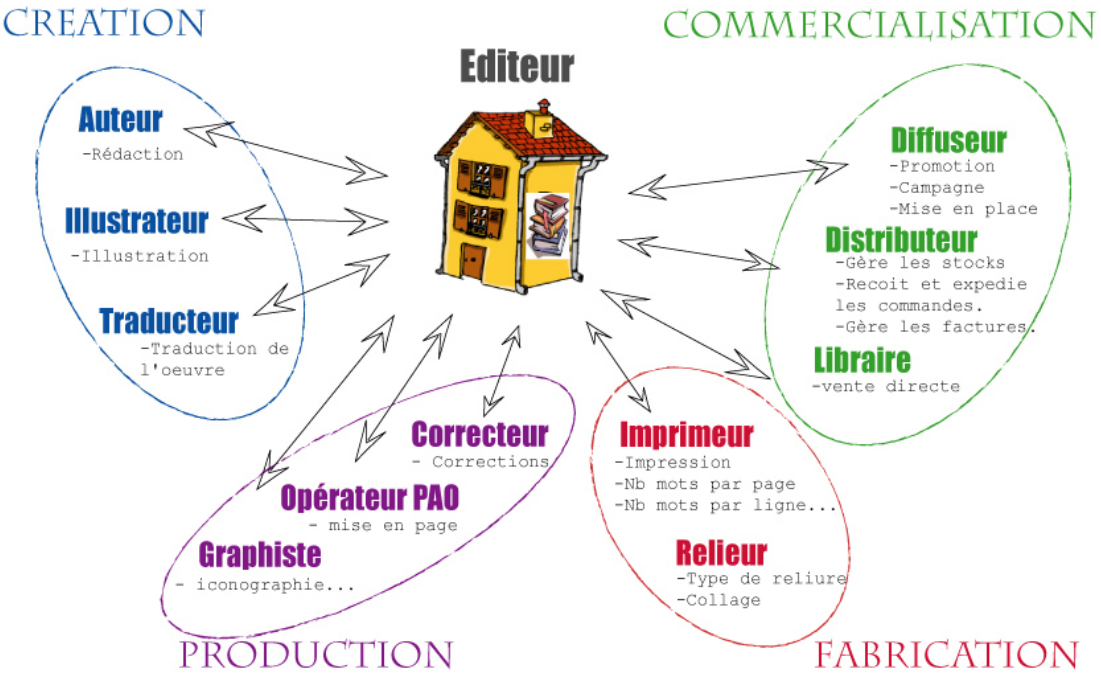

Fig. 1.1 – Chaîne de fabrication d'un livre

L'évolution des systhèmes d'information et de diffusion éléctronique ainsi que l'emergence de nouvelles normes dans le domaine de l'édition permettent une information plus "riche" sur le suivi d'un ouvrage.

Largement influencé par les libraires en ligne qui veulent exploiter cette possibilité, le monde du livre est sur le point d'aller au-delà de la transmission de données purement bibliographiques. Ces données "riches" peuvent comprendre des informations sur :

- un type de livre ;
- différents types d'identifiants (ISBN,ENA.UCC-13,UPC...) ;
- son titre ;
- son résumé ;
- son éditeur ;
- son imprimeur ;
- sa mise en page ;
- son(ses) auteur(s) ;
- sa disponibilité ;
- ses prix ;
- sa date d'édition ;
- ...

La norme d'échanges ONIX [4] a donc été créée pour répondre à cette demande.

1.2 onix ?

ONIX est une norme internationale pour la diffusion de méta-données riches concernant des livres et d'autres documents utilisés par les bibliothèques et les éditeurs. Ses principes directeurs [guidelines] comprennent des spécifications de contenu, d'éléments de données, d'étiquettes et de listes de codes ainsi qu'une DTD XML.

ONIX prend en compte :

- les renseignements bibliographiques complets ;
- du texte : descriptions, comptes rendus, biographies, extraits ;
- des images : couvertures, onglets, photos des auteurs...
- de l'audio et de la vidéo, des liens vers des sites Web ;
- les droits territoriaux ;
- les prix et la disponibilité sur différents marchés ;
- de l'information sur les campagnes promotionnelles ;
- ...

ONIX est développé et géré par EDItEUR.org[5], dirigé par un comité de pilotage EDItEUR/ONIX International qui contrôle la politique et les priorités. BSIG[6] et BIC[7] ainsi que d'autres groupes nationaux ou régionaux y ont des représentants. Une équipe technique ONIX est responsable du développement et de la documentation et EDItEUR gère une liste de discussion ouverte à tous les utilisateurs et aux futurs utilisateurs.

[4] Online Information Exchange
[5] http ://www.editeur.org/.
[6] Book Industry Study Group.
[7] Book Industry Communication.

En résumé, ONIX est une norme qui peut accroître et améliorer le flux de l'information bibliographique et de l'information sur d'autres produits (CD audio, DVD, disquettes, microfilms, imprimé divers ...) d'un bout à l'autre de la chaîne de production. ONIX offre aussi au monde de l'édition et à celui des bibliothèques une bonne occasion de collaborer.

1.3 Besoins des Presses universitaires de Caen

Les Presses universitaires de Caen communiquent avec leurs collaborateurs à l'aide de fichiers XML conformes à la DTD ONIX. Ainsi, si l'un d'entre eux souhaite avoir des informations sur un livre, il lui suffit de se connecter au site Internet des Presses universitaires de Caen et de récupérer le fichier correspondant. S'il s'aperçoit d'une erreur le concernant dans l'un de ces fichiers, il doit, pour la corriger, en informer les Presses universitaires de Caen qui effectuerons les modifications dans leur base de données et refabriqueront un fichier correct. Ceci est assez fastidieux et n'est pas forcément immédiat. De plus, ils ne possèdent pas de système centralisé qui leur permetterait de gérer efficacement leurs données éditoriales. Par conséquent, la mise à jour du catalogue implique un travail supplémentaire de collecte de données éparses.

Bien qu'à l'origine ONIX ait été créée pour échanger de nombreuses données bibliographiques sous forme de messages, il est courant de la détourner de sa fonction première en créant une "base de données maison". Il sera ainsi possible de stocker en un seul endroit toutes les données et utiliser un outil pour les administrer.

Les Presses universitaires de Caen souhaitent profiter de cette possibilité et permettre ainsi à chacun de ses collaborateurs de pouvoir modifier directement et simplement ses propres informations.

Il sera donc nécessaire de créer différents interfaces de saisie correspondant chacun à un type d'intervention dans la chaîne de fabrication d'un livre.

Chapitre 2

Analyse

2.1 Définition des besoins

Actuellement, les Presses universitaires de Caen assurent le suivi éditorial d'une œuvre de manière informelle. C'est-à-dire que chaque service participe à ce suivi à sa façon : fichiers WORD ou EXCEL pour les un, site Internet pour les autres, etc. . .Cela peut poser de nombreux problèmes, surtout pour la maintenance et la mise à jour du catalogue.

Pour éditer le catalogue, les utilisateurs utilisent l'outil *FrameMaker*[1] qui permet l'import de données XML.

Comme nous l'avons vu précédemment, la norme ONIX, qui répond bien aux attentes en matière d'édition, est utilisée pour la création de ces données.

[1] un outil de mise en page.

La solution actuelle pour la gestion des données éditoriales et l'édition du catalogue est décrite dans le schéma suivant :

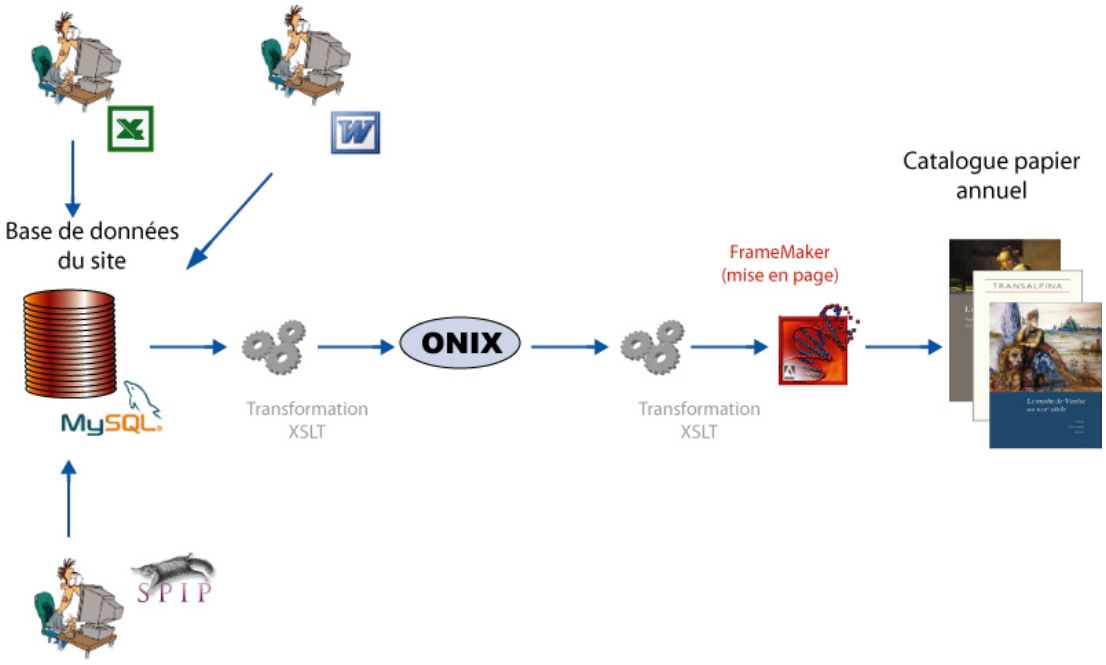

Fig. 2.1 – Solution actuelle

Explications : l'utilisateur rentre les données éditoriales (nom d'auteur, titre, nombre de pages, résumé...) via le CMS[2] SPIP.

Un outil de conversion *maison* permet d'extraire les données de la base du site Internet et de les transformer en XML.

Un autre outil retravaille les fichiers obtenus de façon à ce qu'il soit accepté par *framemaker*.

Le catalogue est enfin mis en page puis publié via ce dernier.

La maintenance de la base de données du catalogue est à ce jour opérationnelle mais pas pratique car il faut rechercher les données qui ne sont pas toutes stockées au même endroit (dans des fichiers word, pour les uns, dans des fichier excel pour les autres ou bien sur la base du site Internet). De plus, la base de données de SPIP n'est pas conçue spécialement pour gérer des données éditoriales, mais pour mettre du contenu en ligne sous forme de rubrique et d'article. Par conséquent, le travail est fait à l'envers (on part de l'ouvrage qui est en ligne pour le retransformer en données éditoriales).

Afin de pallier à ce genre de problèmes, les Presses universitaires de Caen envisagent la création d'un système qui permettra d'assurer un suivi éditorial plus centralisé du dépôt du manuscrit jusqu'à la diffusion.

[2]Content Management System (un système de gestion de contenu, manière rapide et automatique de gérer un site Internet).

Le schéma suivant nous donne une idée sur l'architecture du projet envisagée :

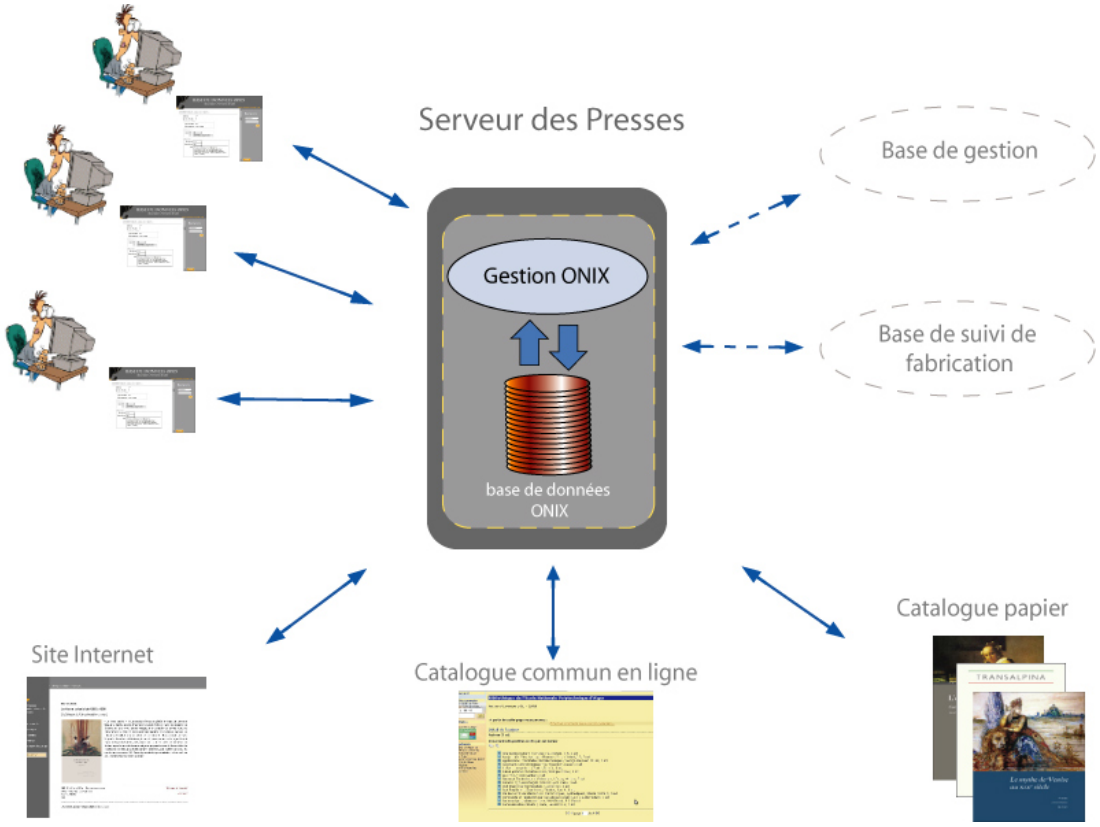

Fig. 2.2 – Solution envisagée

Explications : l'idée est donc de centraliser les données en un seul endroit pour que chaque membre des Presses universitaires de Caen puisse y accéder avec des restrictions en fonction de sa place dans la chaîne de production.

Il s'agit ensuite de créer une interface avec cette base de données centrale qui permettra aux utilisateurs de manipuler des données xml onix en fonction de leur profil.

Cet ensemble forme ce qu'on appellera le *noyau* onix.

Ce noyau sera le point central du système et permettra :

- de mettre à jour le site Internet des Presses universitaires de Caen ;
- de répondre aux interrogations distantes ;
- l'édition du catalogue papier d'une manière plus simple ;
- une interaction avec une base de suivi de fabrication ;
- une interaction avec une base de gestion ;
- et bien plus...

2.2 Architecture complète du projet

Notre projet sera architecturé de la manière suivante :

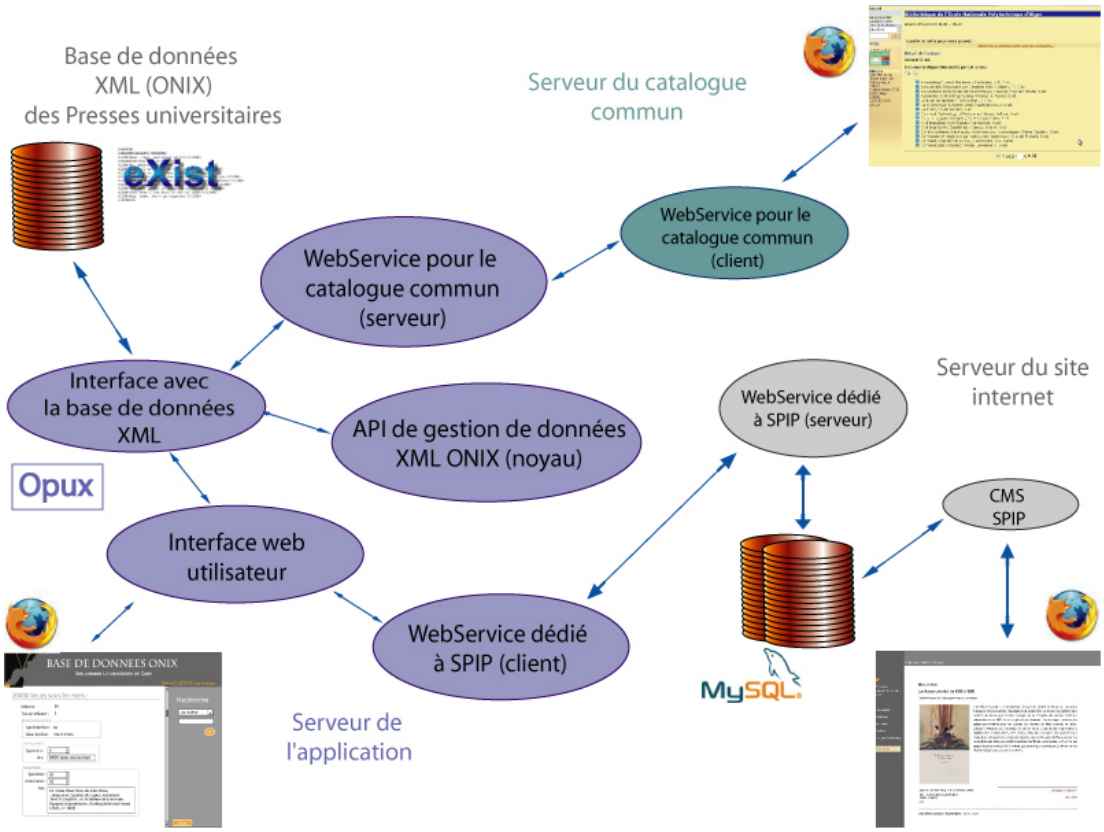

Fig. 2.3 – Architecture de l'ensemble du projet

Sur ce schéma communiquent trois serveurs.

2.2.1 Le serveur de l'application

Il contient le noyau ONIX qui est composé de cinq modules interagissant ensemble :

- une interface avec l'utilisateur ;
- un outil pour manipuler les données ;
- une interface avec la base de données ;
- un Web Service qui permet la mise à jour du site Internet ;
- un Web Service qui répond aux requêtes distantes.

L'interface avec l'utilisateur Elle va permettre de prendre en compte les actions de celui-ci. Elle représente le côté visuel du noyau. Avec l'aide de celle-ci l'utilisateur pourra interroger la base, modifier des ouvrages, commander la mise à jour du site Internet.

Un outil pour manipuler les données Il va permettre l'exploitation de données XML au format ONIX. C'est cet outil qui va fournir une représentation des éléments XML, compréhensible par les autres modules.

L'interface avec la base de données Elle assure la connexion avec la base de données et fournit un moteur de recherche. Celui-ci utilise l'API décrite ci-dessus pour communiquer avec l'utilisateur via l'interface qui lui est dédiée. L'utilisateur, grâce à ce moteur de recherche, peut récupérer une liste d'ouvrages en lançant une recherche sur :

- une notification (s'il est publié, en attente,etc...) ;
- une référence d'ouvrage ;
- un ISBN ;
- un nom d'auteur ;
- les mots d'un titre.

Un Web Service qui permet la mise à jour du site Internet Le site des Presses universitaires de Caen est géré par le CMS SPIP[3]. L'utilisateur pourra, via ce Web Service, ajouter, modifier ou supprimer un ouvrage qui est publié sur le site s'il dispose des droits nécéssaires.

Un Web Service qui répond aux requêtes distantes Il permet l'accession à certaines données publiques des Presses universitaires de Caen (titre, auteurs, url) depuis l'extérieur. Ainsi, si d'autres Presses universitaires possèdent un tel Web Service, il sera possible de les interroger tour à tour pour former un catalogue commun qui sera accéssible depuis le site Internet de l'AERES [4] par exemple.

[3]Système de Publication Pour l'Internet - http ://www.spip.net/.

[4]Association des Éditeurs de la Recherche et de l'Enseignement Supérieur.

2.2.2 Le serveur du moteur de recherche commun

Le serveur du moteur de recherche commun hébergera un Web Service qui sera client de chacun des autres Web Service décrit ci-dessus. Il mettra en forme le résultat de chaque serveur interrogé et le mettra en ligne.

2.2.3 Le serveur du site Internet

C'est dans la base de données du site Internet des Presses universitaires de Caen, hébergée sur ce serveur, que sont stockées la plupart des informations éditoriales.

Cette base de données est utilisée par SPIP qui permet la manipulation d'un tel type de données et de les mettre en forme pour les publier sur le site.

Le serveur est composé d'un Web Service dédié à SPIP qui permettra les modifications de la base de données à distance.

2.3 Un outil permettant de manipuler des données onix

2.3.1 Gestion des utilisateurs

L'analyse sur la gestion des utilisateurs a été faite à l'occasion d'un projet précédent et a subi peu de modifications. En confrontant l'analyse précédente (qui a été faite dans le cadre d'un projet individuel) aux besoins des utilisateurs dans un milieu professionnel, la possibilité d'interagir avec le site Internet des Presses universitaires de Caen a été envisagée.

Le diagramme de cas d'utilisation suivant nous montre la façon dont les utilisateurs pourront agir sur le système.

Diagramme de cas d'utilisation

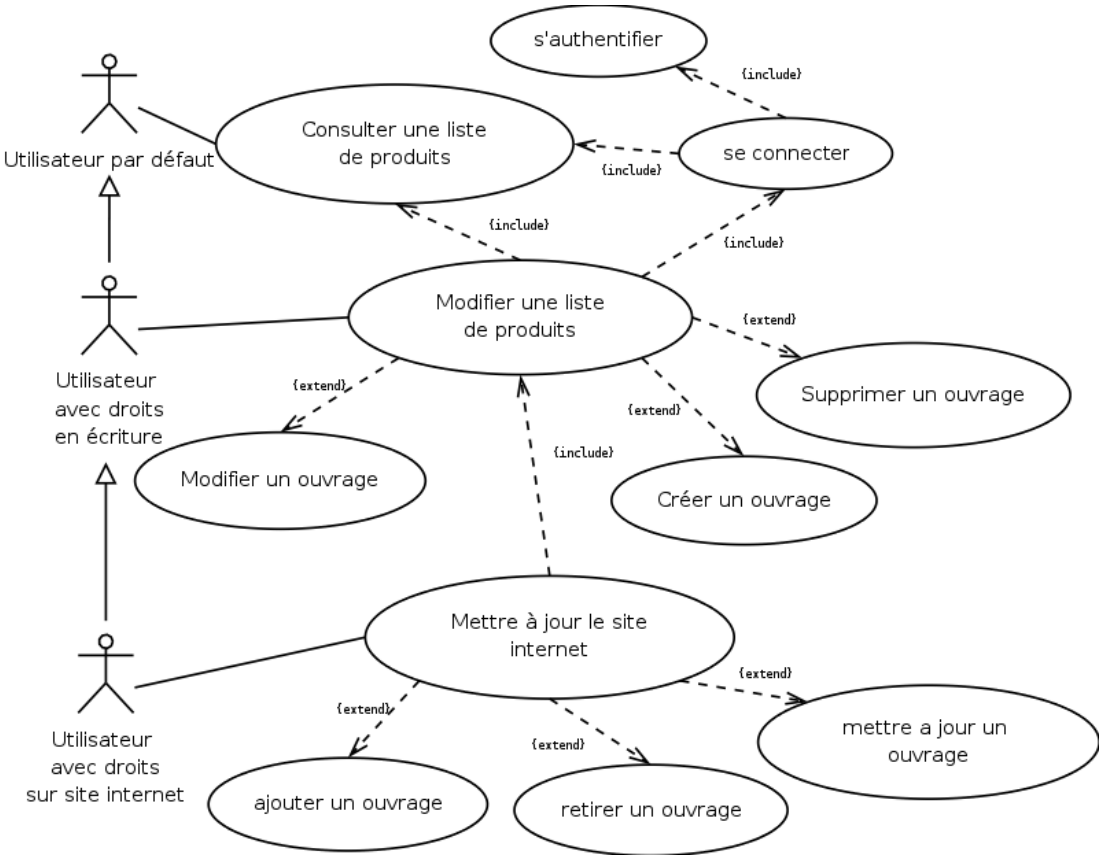

Fig. 2.4 – Cas utilisateur

Explications : pour pouvoir consulter un catalogue, n'importe quel utilisateur devra se connecter et s'authentifier. Ceci permettra la récupération de ses droits. Un utilisateur avec des droits en écriture pourra :

- modifier un ouvrage dans la base de données ;
- ajouter un nouvel ouvrage ;
- supprimer un ouvrage.

Un utilisateur avec des droits d'administration du site Internet :

- insérer un nouvel ouvrage ;
- mettre à jour un ouvrage ;
- retirer un ouvrage du site.

Une interface adaptée aux droits

L'acces aux données en fonction des droits de l'utilisateur passe par plusieurs étapes :

1. L'utilisateur entre son *login* et mot de passe, puis reçoit un formulaire de recherche.
2. Il interroge ensuite la base sur un auteur, un titre, une référence...). Il recevra une liste d'ouvrages qu'il pourra modifier en fonction de ses droits.
3. Enfin il recevra pour chaque ouvrage consulté, un formulaire contenant les champs qu'il a le droit de lire et ceux qu'il peut modifier. Il disposera également d'une "console " pour apporter des modifications au site Internet, s'il en a le droit.

Le schéma suivant nous montre un exemple d'utilisateur connecté, avec des droits sur certains champs :

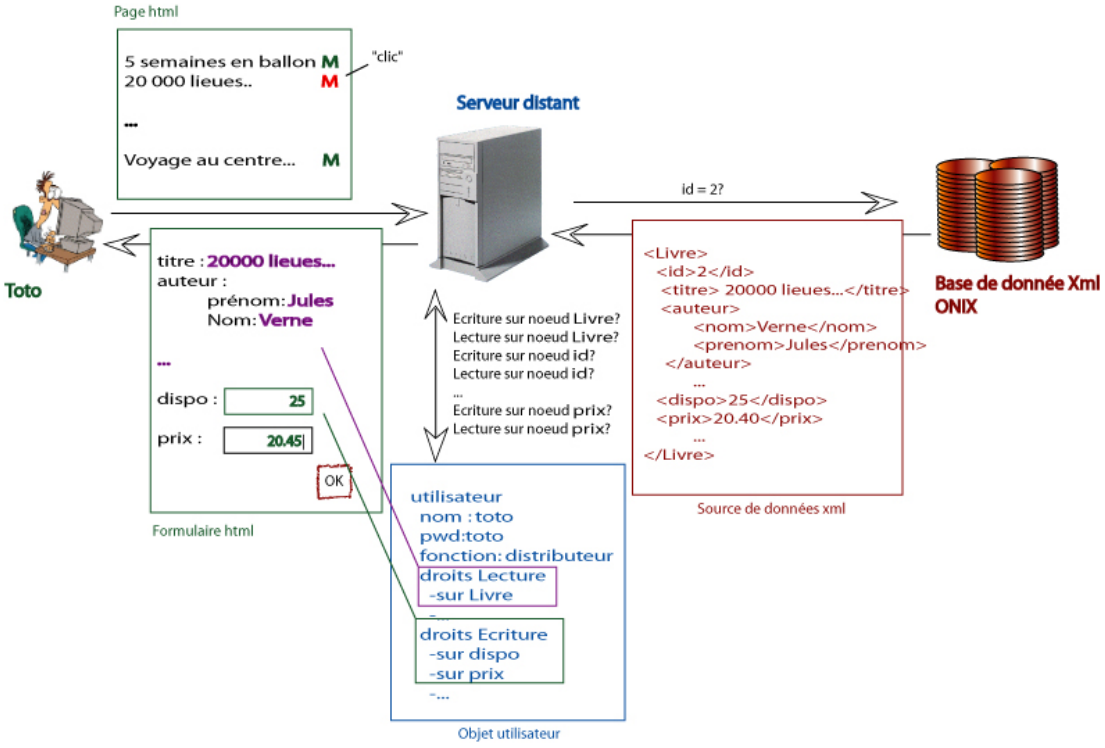

Fig. 2.5 – Modification du produit suivant les droits

Explications :

- L'utilisateur demande les données d'un ouvrage qu'il vient de choisir.
- Le serveur interroge la base sur la référence de l'ouvrage demandé.
- Et pour chaque champ de cet ouvrage :
  - si l'utilisateur a des droits en écriture sur ce champ, il affiche celui-ci de manière à ce qu'il puisse le modifier.

  - si l'utilisateur n'a que des droits en lecture sur ce champ, le serveur affiche seulement celui-ci.
  - sinon il ne l'affiche pas.
- Le serveur envoie enfin le formulaire adapté aux droits de cet utilisateur.

L'utilisateur de notre exemple a peu de chance d'avoir de droits sur la mise à jour du site Internet pour cet ouvrage dans la mesure où ses droits sont restreints. Cependant, cela reste possible si l'administrateur le souhaite (rien n'empêche en effet notre utilisateur de pouvoir mettre à jour l'ouvrage qui est en ligne avec le peu de données qu'il est en droit de modifier).

Il découle de cette analyse la nécessité de trouver un moyen d'attribuer des droits à différents utilisateurs. XML a été choisi pour cette nécessité et en particulier pour pouvoir simplifier, par la suite, la mise en place d'un outil d'édition graphique de ces droits.

Définition d'un fichier xml des utilisateurs

Que peut-on dire d'un utilisateur :

- Un utilisateur à besoin d'un *login* et d'un mot de passe qui lui permettront de se connecter ;
- on peut avoir besoin également de quelques informations sur l'utilisateur, notamment sa fonction ;
- il nous faut savoir ensuite ce qu'il a le droit de lire et d'écrire dans la base de données ;
- il nous faut enfin savoir s'il pourra mettre à jour le site Internet (plutôt réservé au chargé de fabrication et bien sûr à l'administrateur du site).

Le schéma suivant permet de décrire ces besoins. On l'utilisera pour élaborer un fichier XML qui sera interrogé par le serveur.

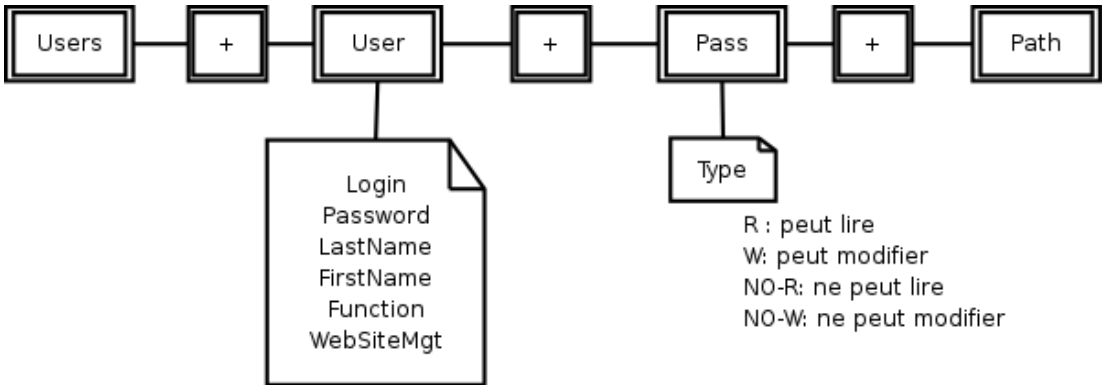

Fig. 2.6 – Structure d'un profil utilisateur

Le fichier xml des utilisateurs aura un ou plusieurs utilisateurs, chacun ayant pour attributs *login, password, firstname, lastname, function* et *webSiteMgt*. L'attribut *function* permet de savoir la place qu'occupe l'utilisateur dans la chaîne de fabrication d'un livre et l'attribut *webSiteMgt* s'il peut ou non modifier les ouvrages en ligne sur le site des Presses universitaires de Caen .

Chaque utilisateur possède un ou plusieurs droits (*pass* dont le *type* pourra être :

- **R** pour des droits en lecture ;
- **W** pour des droits en écriture ;
- **NO-R** pour qu'il n'ait pas de droits en lecture ;
- **NO-W** pour qu'il n'ait pas de droits en écriture.

Enfin, chaque *pass* sera composé d'un ou plusieurs chemin Xpath (*path*).

La notion de non-droit n'existait pas lors d'une précédente analyse. Elle a été rajoutée pour des raisons d'ergonomie. En effet, si un utilisateur n'a que deux champs interdits, on n'aura que deux chemins Xpath à ajouter à son profil en "NO-R" ou "NO-W".

L'attiribut *webSiteMgt* a également été intégré pour les raisons citées précédemment.

2.3.2 Manipulation de données onix

Analyse d'un fichier xml fourni par les Presses universitaires de Caen

Bien que la dtd onix possède plus de 250 éléments, les Presses universitaires de Caen n'en utilisent qu'une quarantaine. L'étude de ces fichiers et l'extraction des différents éléments onix utilisés par les Presses universitaires de Caen ont constitué une grosse partie de l'analyse faite lors du projet individuel.

En confrontant la confrontant aux besoins des utilisateurs, il a fallut la reprendre point par point.

Voici un aperçu d'un de ces fichiers :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<ONIXMessage>
<Header>
  <FromCompany>Presses universitaires de Caen</FromCompany>
  <SentDate>20060119</SentDate>
</Header>
<Product>
  <RecordReference>31</RecordReference>
  <NotificationType>03</NotificationType>
  <ProductIdentifier>
    <ProductIDType>02</ProductIDType>
    <IDValue>2905461411</IDValue>
  </ProductIdentifier>
  <ProductIdentifier>...</ProductIdentifier>
  <ProductForm>BC</ProductForm>
  ...
  <Title>
    <TitleType>01</TitleType>
    <TitleText>Principes du droit budgetaire...</TitleText>
  </Title>
  <NumberOfPages>356</NumberOfPages>
  <OtherText>
    <TextFormat>02</TextFormat>
    <Text">Le principe de l'unité du budget...</Text>
  </OtherText>
  <Publisher>
    <PublishingRole>01</PublishingRole>
    <PublisherName>PUC</PublisherName>
  </Publisher>
  <PublicationDate>1989</PublicationDate>
  <Measure>
    <MeasureTypeCode>02</MeasureTypeCode>
    <Measurement>14</Measurement>
    <MeasureUnitCode>cm</MeasureUnitCode>
  </Measure>
  <Measure>...</Measure>
  <SupplyDetail>
    <SupplierName>Presses U de Caen</SupplierName>
    <ProductAvailability>IP</ProductAvailability>
    <Price>
      <PriceTypeCode>02</PriceTypeCode>
      <PriceAmount>22,87</PriceAmount>
    </Price>
  </SupplyDetail>
</Product>
</OnixMessage>
```

Extraction des différents éléments

De ces fichiers ont pu être retirés une quarantaine d'éléments qui sont la base de notre API. Chacun de ces éléments possède des contraintes liées à la DTD ONIX fournie par EDItEUR. Ces contraintes ont été étudiées lors de l'analyse précédente ce qui a permis d'obtenir le diagramme suivant (on pourra trouver les autres en annexe) :

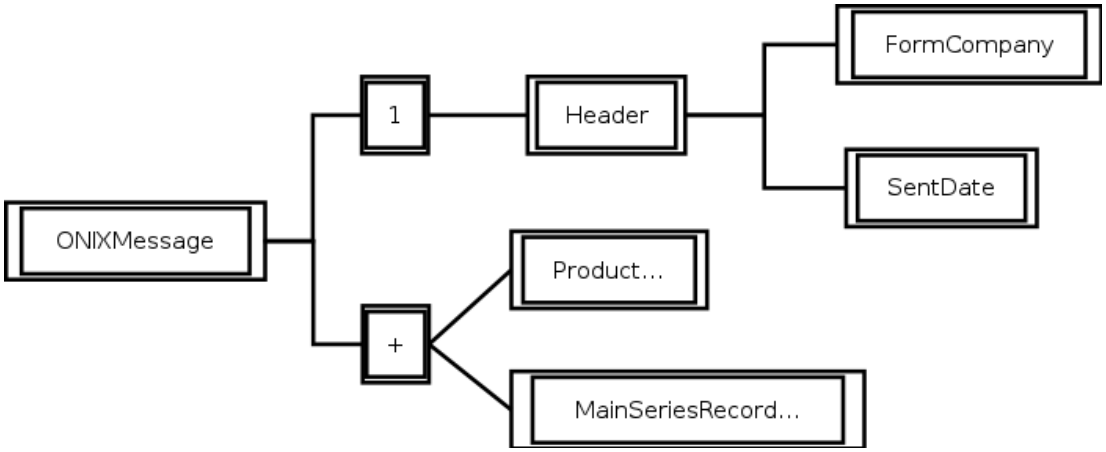

Fig. 2.7 – Structure d'un message ONIX

Formalisme utilisé :
- 1 ou rien : l'élément est obligatoire ;
- + : il y a au moins 1 élément ;
- * : il y a 0 ou plusieurs éléments ;
- ? : cet élément n'est pas obligatoire ;
- ... : se décompose en sous arbre.

Comment lire cet arbre ? On dira par exemple qu'un *OnixMessage* contient obligatoirement un *Header*, un ou plusieurs *Product* et/ou un ou plusieurs *MainSeriesRecord*. Un *Header* est composé obligatoirement d'un *FormCompany* et d'un *SentDate*.

Il s'agit ici de l'arbre principal. On pourra trouver les arbres détaillés de *Product* et *MainSeriesRecord* dans les annexes.

Ces arbres constituent une bonne base de travail et facilitent le passage à une modélisation objet (UML) des éléments en présence.

Définition d'un diagramme de classes uml suivant cet arbre

Modélisation uml d'un message onix La figure 2.7 représente le squelette d'un message onix . Que peut on dire de l'élément *ONIXMessage* ?

- Qu'il est composé de plusieurs éléments du même type (*Header, Product, MainSeriesRecord*). C'est donc un élément complexe ;
- Les éléments *FormCompany* et *SentDate* ne possèdent pas d'autre élément. Ce sont donc des éléments simples qui possèdent une valeur ;
- qu'un élément complexe peut contenir des éléments simples et/ou des éléments complexes.

On peut modéliser cette analyse de la manière suivante :

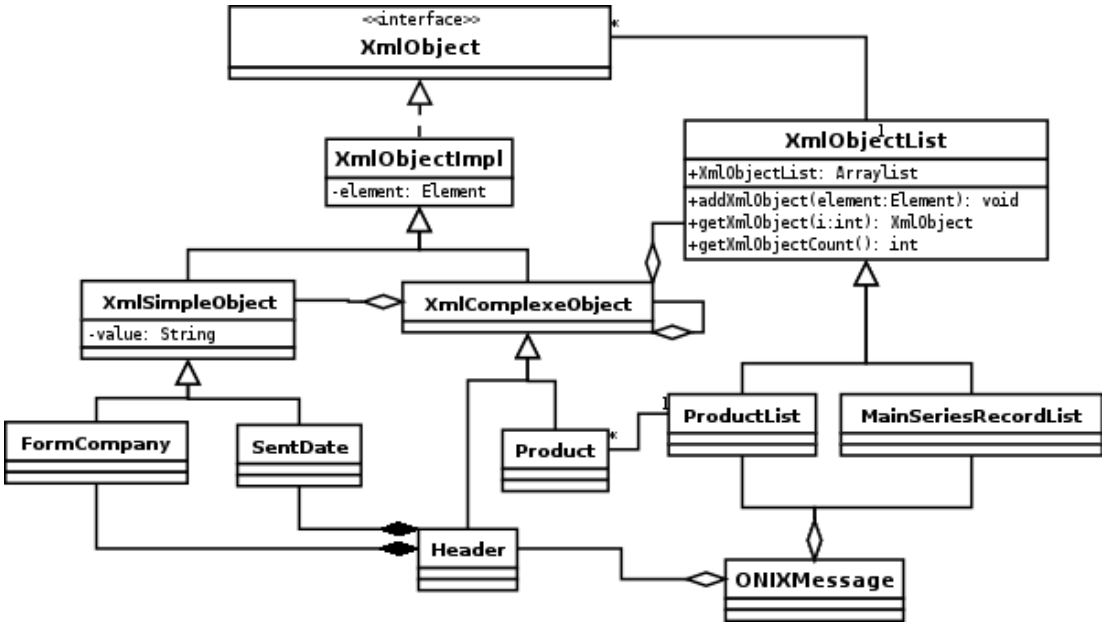

Fig. 2.8 – Diagramme de classes d'un message xml

Comme nous sommes dans un environnement xml, chaque classe devra hériter d'une classe *XmlComplexeObject* pour les éléments complexes et d'une classe *XmlSimpleObject* pour les simples, qui feront la correspondance entre un élément xml et un objet de notre application. Ces classes hériteront d'une classe plus générique (*XmlObjectImpl*) qui implémentera une interface que l'on appellera *XmlObject*.

Cette interface permettra d'assurer par exemple la correspondance entre un objet *header* et sa représentation xml, comme le montre la figure suivante :

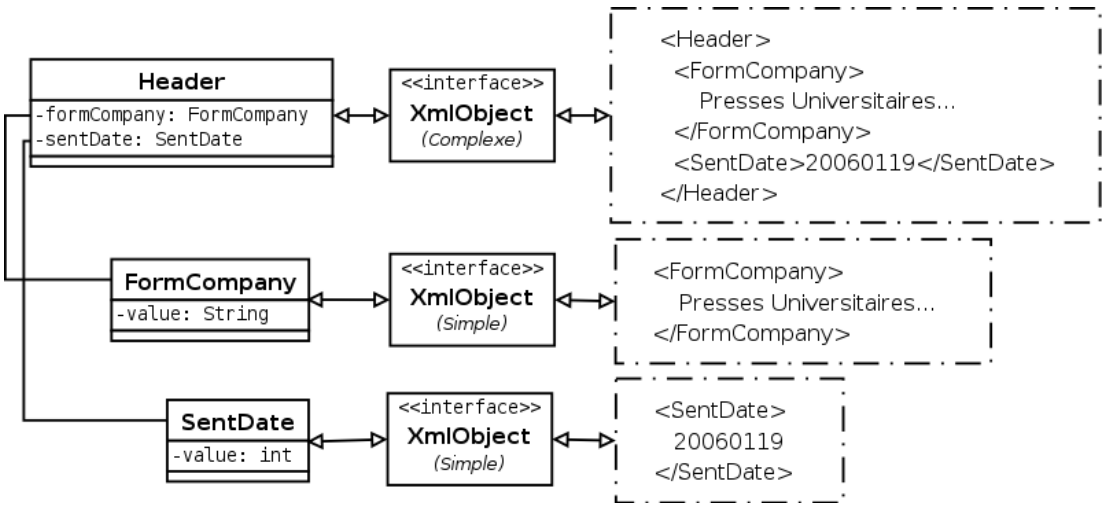

Fig. 2.9 – Exemple d'interface onix

Pour finir, on souhaite dédier cette application à la norme onix. On va donc, pour ce faire, créer une super classe *OnixObject* qui héritera de la classe générique *XmlObjectImpl* implémentant l'interface *XmlObject* ainsi que deux sous classes *OnixSimpleObject* et *OnixComplexeObject* qui hériteront de cette dernière comme le montre la modélisation suivante :

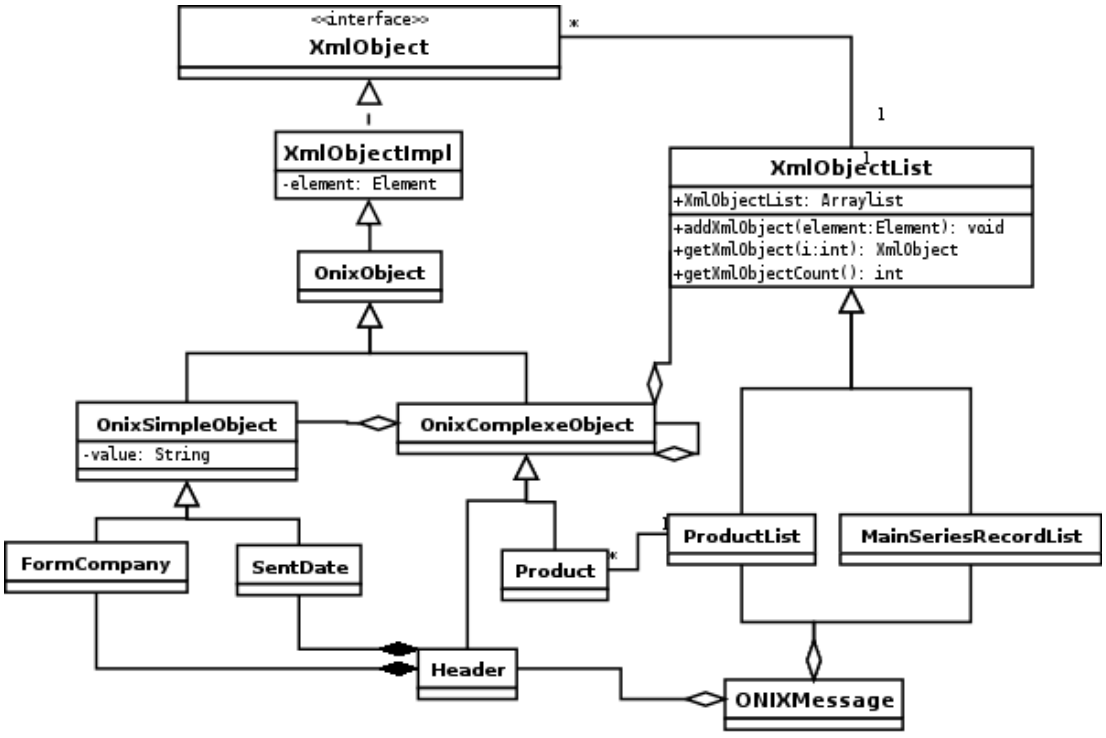

Fig. 2.10 – Diagramme de classes d'un message onix

On trouvera un exemple plus complet sur les *MainSeriesRecord* dans les annexes (le principe étant le même pour les *Product*).

La notion d'éléments simples et d'éléments complexes est apparue lors de la reprise de l'analyse précédente. En effet, j'ai constaté la nécessité de rendre les objets plus génériques lors de l'implémentation de l'interface Web en JSP, notamment lors de l'exploitation des paramètres envoyés par un formulaire HTML de modification d'un ouvrage. Nous en parlerons plus en détails par la suite.

Si on récapitule, la reprise de l'analyse précedente a permis de coller au plus près des besoins des utilisateurs et d'éviter les problèmes de redondance et de généricité qui pouvaient apparaître lors du développement de l'interface Web. Il est désormait possible de créer des *utilisateurs* (chacun possédant une liste de droits), ainsi que des objets représentatifs de la norme ONIX, le tout étant rassemblé en un seul endroit. Les données étant à présent centralisées, nous allons pouvoir répondre à d'autres besoins comme la mise à jour du site internet depuis l'application ou la création d'un moteur de recherche commun.

2.4 Interaction avec le site Web

Les nouvelles technologies permettent maintenant de décentraliser et normaliser les données, ce qui offre de meilleures possibilités de communication. Grâce à celles-ci, il est maintenant possible de mettre à jour un site Internet de façon simple.

Nous allons exploiter cette possibilité afin d'ajouter une fonctionnalité au noyau ONIX qui mettra à jour le site Internet de manière simple.

Cette fonctionnalité va être architecturée de la manière suivante :

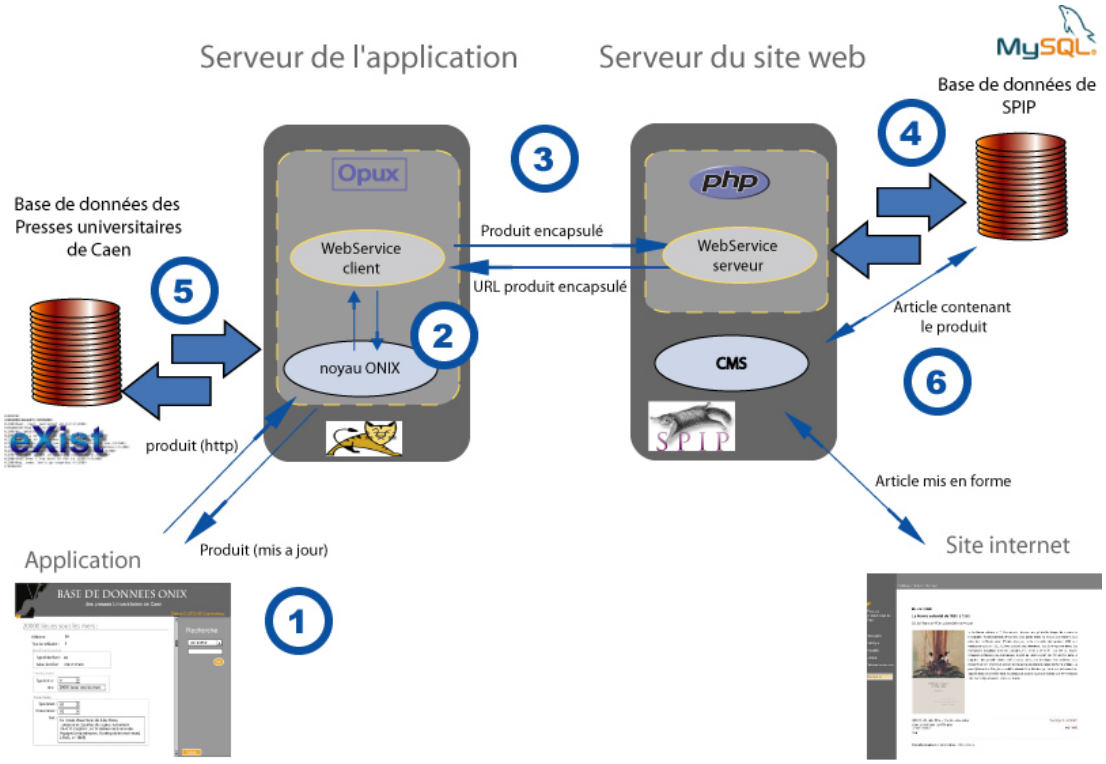

Fig. 2.11 – Architecture du Web Service dédié à SPIP

Explications :

Sur le serveur du site :

- les données éditoriales sont gérées par une base de données MySQL dédiée à SPIP ;
- Le site Internet est fabriqué par SPIP à partir des données de cette base ;
- Un Web Service va permettre de modifier la base de données depuis une application distante.

Sur le serveur de l'application :

- Les données ONIX sont gérées par une base de données XML ;
- Le noyau ONIX (gestion des données, interface base de données, interface utilisateur) est stocké dans une conteneur de *Servlet* (Tomcat par exemple) ;
- Ce noyau possède un module qui sera client du Web Service dédié à SPIP .

La procédure d'ajout d'un ouvrage sur le site est la suivante :

1. le client édite un ouvrage à l'aide de son interface. Après l'avoir modifé, il demande l'insertion de cet ouvrage au site ;
2. le noyau ONIX traite les informations envoyées par le navigateur de l'utilisateur ;
3. il utilise son Web Service client pour envoyer l'ouvrage au Web Service du site ;
4. Celui-ci met à jour la base de données de SPIP, puis renvoie un URL qui correspond à l'emplacement de l'ouvrage sur le site ;
5. le noyau ONIX le récupère par l'intermédiaire de son Web Service client et met la base de données XML à jour ;
6. l'ouvrage est maintenant en ligne et à la bonne place dans le catalogue.

Le principe est le même pour la suppression et la mise à jour.

Nous avons maintenant tous les éléments en main pour maintenir à jour les données éditoriales et les publier sur un site Internet. Plusieurs partenaires des Presses universitaires de Caen [5] souhaitent travailler en commun avec celles-ci. Dans la mesure où leur normes d'échange est la même (ONIX), ces différents partenaires pourront utiliser notre application. Ainsi, pour qu'il soit possible de créer un catalogue commun, l'application devra répondre à des requêtes distantes (par exemple, donne moi tous les livre dont l'auteur est "Verne"). Un moteur de recherche pourra alors venir interroger chaque presses universitaires munie de l'application et afficher les ouvrages correspondants avec un lien vers leur propriétaire.

2.5 Moteur de recherche pour un moteur de recherche commun

Cette fonctionnalité interessera surement l'AERES [6]. Le moteur de recherche sera alors ébergé sur son site.

[5] Les presses universitaires d'autres régions.

[6] Association des Éditeurs de la Recherche et de l'Enseignement Supérieur.

Le schéma suivant nous montre le principe de cette fonctionnalité :

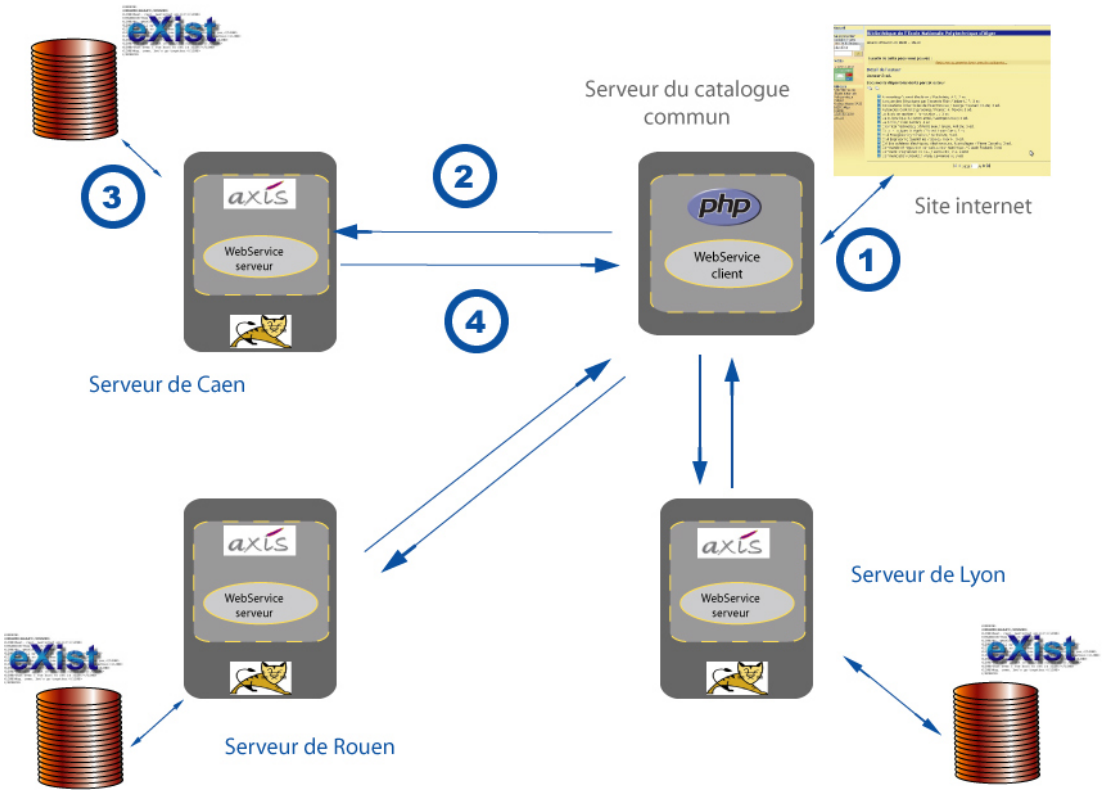

Fig. 2.12 – Principes du moteur de recherche commun

Le serveur du moteur de recherche commun utilise un Web Service client des différent Web Service de chacune des Presses universitaires équipées d'une base de donnée eXist et d'un Web Service qui répond aux demandes de recherches sur cette base.

La procédure d'interrogation de chaque serveur distant est la suivante :

1. un internaute connecté au site du moteur de recherche effectue une demande ;
2. le Web Service de celui-ci interroge tour a tour les différents serveurs distants ;
3. chacun de ces serveurs interroge leur base de données et met en forme le resultat ;
4. chacun des résultats retournés sera mis en forme par le Web Service du moteur de recherche et mis à la disposition de l'internaute.

Chapitre 3

Développement

3.1 Principes généraux, environnement de développement

Il ressort différents besoins de cette analyse. Le plus important est celui de centraliser les données et le moyen de les mettre à jour. Ce qui permettra par la suite d'ajouter au fur et à mesure à ce noyau, des fonctionnalités qui répondront à de nouveaux besoins. Il est donc important que l'application soit évolutive et ce, de manière le plus simple possible.

3.1.1 Architecture modulaire

La meilleure façon de répondre à ce besoin et de développer l'application selon une architecture modulaire. De cette manière, il est plus aisé d'ajouter ou de retirer (voir de décentraliser) certains modules.

Si l'on regarde la figure 2.3 dans la partie analyse, on peut séparer l'architecture en deux partie, une partie local (noyau central) et une partie distante (les différents Web Services).

La partie locale se découpe en 5 modules, ce qui donnera 5 mini-projets chacun composés de tous les packages nécessaires à son fonctionnement :

- l'interface avec l'utilisateur ;
- l'API pour manipuler les données ;
- l'interface avec la base de données ;
- le Web Service qui permet la mise à jour du site Internet ;
- le Web Service qui répond aux requêtes distantes.

La partie distante il y a deux modules :

- le Web Service dédié à SPIP ;
- le Web Service client du Web Service de l'application pour le catalogue.

La compléxité de mise en œuvre du noyau justifie qu'il faille mettre en place une architecture lourde. C'est pour cette raison qu'il sera implémenté dans un environnement J2EE. De plus cette architecture est basée sur un langage objet, est idéale pour une programmation modulaire et multi-plateforme.

Interface avec utilisateur

L'interface avec l'utilisateur représente la partie visible du noyau. Il reste à savoir comment seront pris en compte les actions et les saisies de l'utilisateur et qui va décider de la marche à suivre. C'est le rôle du contrôleur de Struts. Le contrôleur est implémenté en deux parties : une partie Java (la classe Action) et une partie Struts. L'**Action** reçoit les saisies utilisateur, coordonne l'accès au stockage persistant, met en oeuvre la logique métier et décide de la page JSP qui sera affichée à l'utilisateur. Struts, qui est configuré à l'aide du fichier XML struts-config.xml[1], s'occupe de la transmission à la page suivante.

Une api pour manipuler les données

Ce module est un peu le cœur du noyau. C'est lui qui permet la manipulation des données XML et garde leur représentation en mémoire. Il assure également les contrôles d'intégrité pour que les données restent conformes à la DTD ONIX.

Interface avec une base de données xml

Ce module sert de pont entre l'API décrite ci-dessus et la base de données eXist. Il possède entre autre deux objet principaux :

- un objet *connexion* ;
- un objet *moteur de recherche*.

L'interface et les différents Web Service utiliseront ce module couplé à l'API pour ce connecter, interroger et modifier la base.

Un Web Service qui permet la mise à jour du site Internet

Ce module permet la communication avec le Web Service dédié à SPIP. Il dispose d'une classe *Facade* qui permet d'utiliser simplement les méthodes propsées par le Web serveur.

Un Web Service qui répond aux requêtes distantes

Nous disposons d'un conteneur de servlet **Tomcat** qui héberge notre application et qui peut également héberger un conteneur de Web Service. Ainsi nous utilisont **Axis** pour héberger notre Web Service car celui-ci nous offre des outils très puissants d'aide à la création de Web Service.

La création des Web Services de la partie distante est par contre plus aisée et ne nécessite pas une technologie aussi lourde que la précédente. C'est pour cela qu'un environnement PHP a été choisi pour les créer. De plus, SPIP est déjà programmé dans cet environnement.

[1] cf annexes.

Le Web Service dédié à spip

Pour ce qui est de la communication entre les Web Services, on utilise le protocol SOAP, dont PHP5 intègre les fonctions.

- *insertProduct(String xmlProductString)*, méthode qui parse la donnée XML en paramètre et fabrique une requête (SQL) d'insertion dans la base ;
- *updateProduct(String xmlProductString)*, même procédé que ci-dessus mais pour une requête de mise à jour ;
- *removeProduct(String urlProduct)*, méthode qui récupère la référence de l'ouvrage contenue dans l'url et fabrique une requête de suppression sur le numéro récupéré ;
- *getRubriques()*, méthode que fourni une liste des rubriques disponibles afin que l'on puisse en choisir une pour y insérer un ouvrage.

Le Web Service client du Web Service de l'application pour le moteur de recherche commun

Également implémenté en PHP, il dispose d'une liste de serveur qui possèdent l'application et appelle les méthodes de recherches fournies par chacun de ces serveurs. Il met les résultats en forme à l'aide d'une feuille de style XSLT.

Étant donné que tous ces modules doivent échanger des informations entre eux, que notre application est basée sur la norme ONIX et que celle-ci est totalement basée sur le langage XML, ce langage sera le socle commun à toutes les fonctionnalités de l'application.

3.1.2 Une programmation "full xml"

Le fait d'axer la programmation sur ce socle offre bien des avantages :

- pas besoin de fabriquer de l'XML, puisque les informations sont déjà en XML dans la base ;
- cela permet la communication avec des modules distants indépendemment de leurs language de programmation ;
- cela offre toutes les possibilités fournies par des normes telles que XPATH, XSL...
- les DTD permettent une certaine assurance sur les données que l'on souhaite manipuler, évitant ainsi des erreurs ;
- et bien d'autres avantages...

Gestion des droits

La gestion des droits ce fait à l'aide d'un fichier XML structuré comme l'indique la figure 2.6 dans la partie analyse. Cette manière de procéder nous permet de respecter notre axe de programmation, et également de réutiliser l'interface *XmlObject* et ses implémentations avec tous les outils qui vont avec.

Stockage des données

Il existe plusieurs solutions pour stocker des données XML. On peut stocker chaque produit dans un fichier XML et ranger ce genre de fichier dans différents répertoires. Ce qui oblige à créer et mettre à jour un index, ce qui n'est pas très performant et limite les requêtes. Une autre solution, celle qui a été retenue, est d'utiliser une base de données XML tel que eXist[2]. Plusieurs raisons ont motivé ce choix :

- eXist peut s'intégrer très facilement dans une application Java(utilisation de l'API XML :DB[3]) ;
- on peut l'interroger à l'aide de requêtes XPATH ;
- eXist fournit des fonctions adaptées à la recherche textuelle et aux concepts plus orientés base de données ;
- eXist va pouvoir fournir des DOM[4] à partir desquels nos *OnixObject*(voir paragraphe **??**, page **??**) pourront se construire ;
- et surtout, eXist étant une base de donnée native, elle nous permet de rester dans notre axe de programmation.

Rechercher des données

Toutes les requêtes d'interogation de la base de données sont effectuées en XPATH2.0 car eXist fourni des outils puissants de recherche avec ce langage.

Échange de données

L'échange de données entre tout le noyau de l'application et les modules distant est bien sûr effectué en XML encapsulé par SOAP bien souvent.

configurabilité de l'application

L'application utilise également ce langage pour ces différents fichiers de configuration.

- Un fichier des listes de codes ONIX (disponible en annexes) fourni par EDiTEUR, permet de faciliter la saisie des données (ce qui évite par exemple, d'avoir à connaître un code de type de titre d'un ouvrage par coeur,etc...). L'application utilise donc ce fichier pour chaque liste de codes. Ceci à l'avantage de permettre de modifier les listes sans avoir à recompiler.
- De la même façon, l'application utilise un fichier XML qui contient un produit par défaut (*defaultProduct.xml*) pour proposer un ouvrage pré-rempli quand l'utilisateur souhaite en créer un nouveau. Ceci offre le même type d'avantage que le précédent.

[2]http ://exist-db.org/.
[3]http ://www.xmldb.org/.
[4]http ://www.w3.org/DOM/.

3.1.3 Environnement matériel

J'aivais à ma disposition en arrivant un environement Solaris10 sur une SUN et une connexion au réseau. J'ai donc passé quelque temps à étudier ce système et installer tout l'environnement nécessaire à l'application (packaging de SUN, environnement J2EE, Tomcat, eclipse, PHP, MySQL...

3.2 Problèmes rencontrés

Encoding

Le problème qui ne se posait pas vraiment lors de mon projet individuel, mais qu'il a fallu résoudre imperativement après avoir été confronté aux besoins des utilisateur. En effet, les Presses universitaires de Caen en leur qualité d'éditeur pourront être amenées à utiliser des caractères qui ne font pas partie de la norme ISO-8859-1. C'est pour cela que toutes les données doivent être encodées dans une norme universelle (UTF-8).

Ayant un peu laissé de coté ce problème au début en pensant le résoudre rapidemment, celui-ci c'est vite avéré un casse-tête et a nécessité de nombreuses recherches sur internet.

Le problème se posait lors de la récupérations des données envoyées par les formulaires HTML lors de la modification d'un ouvrage.

En effet, du coté où les données étaient saisies (navigateur) l'encoding était bon et du coté ou elles étaient traitées aussi. On ne maîtrisait pas cependant la couche ajoutée par le mécanisme des JSP.

La solution à donc été de filtrer les *Servlet* en créant une classe *EncodingFilter* :

```
import java.io.IOException;

public class EncodingFilter implements javax.servlet.Filter {
  private String encoding;

  public void init(FilterConfig filterConfig) throws
      ServletException {
    this.encoding = filterConfig.getInitParameter("encoding");
  }

  public void doFilter(ServletRequest request, ServletResponse
      response, FilterChain filterChain) throws IOException,
      ServletException {
    request.setCharacterEncoding(encoding);
    filterChain.doFilter(request, response);
  }

  public void destroy() {
  }
}
```

et en la déclarant dans le fichier web.xml :

```
<filter>
  <filter-name>Encoding Filter</filter-name>
  <filter-class>fr.unicaen.mi3.struts.utils.EncodingFilter</filter-
      class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>Encoding Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Souplesse de la dtd onix difficile à implémenter

Les champs de données d'un formulaire qui permettent la modification d'un ouvrage ne sont pas fixes (il ne s'agit pas de mettre ses coordonnées dans des champs obligatoires et de laisser vide ceux qui ne le sont pas comme dans un formulaire traditionnel).

En effet, d'après la DTD ONIX, un ouvrage peut avoir un ou plusieurs auteurs, mais n'est pas obligé d'avoir de détails sur son prix, son nombre de pages, etc. Imaginez la taille du formulaire, si tous les champs possibles devaient y figurer !

De plus, la gestion des droits vient encore compliquer la chose, car un utilisateur peut très bien lire des champs sans pouvoir les modifier.

Le problème se pose lorsque l'on veut récupérer les données d'un formulaire. En effet, on ne peut pas savoir à l'avance ce qu'il nous envoie. Donc, difficile de controler si tel champs est bien remplis, ou récupérer la valeur de tel champs si ont ne sait même pas si celui-ci va figurer dans les données envoyées par le formulaire.

Tout d'abord, clarifions la manière dont l'échange de données se fait :

1. au moment où l'utilisateur clic sur le bouton *valider*, les données sont envoyées par la méthode POST classique.
2. ces données sont récupérées par un objet *HttpServletRequest* qui nous les met à disposition sous forme de tableau associatif *NomDuChamps :valeur*.

Il Faut donc parcourir cette liste pour mettre à jour notre ouvrage. La difficulté vient du fait que, par exemple, un livre peut avoir plusieurs auteurs, mais qu'il n'y a qu'un champs dans la DTD pour le désigner par son nom.

Prenons un exemple : On veut modifier le texte du deuxième titre d'un ouvrage. Il faut que ce champs texte soit nommé différemment de celui du premier titre. Pour résoudre ce problème il a fallu convenir d'une règle de nommage pour

tous les champs. Ce qui permet de les identifier de manière unique. La convention choisie est la suivante :

- *NomDuChamps : position : NomDuFils : PositionDuFils : NomduPetit-Fils...*

Ce qui donne dans le cadre de notre exemple :

- *Title :1 :TitleText*

Pour le choix du nom des champs, il suffit de reprendre ceux de la norme ONIX par soucis d'harmonie.

Suite à la reprise de l'analyse précédente, les objets de l'application sont maintenant assez génériques pour pouvoir être mis à jour à partir d'un nom de champs.

On utilisera les méthodes *getOnixComplexeObjectByTagName(String tagName)*, *getOnixSimpleObjectByTagName(String tagName)* et *setValue("Le chat enterré")* pour assurer la mise à jour d'un ouvrage. On retrouvera ces méthodes en annexe dans le listing de la classe *ModifyProductAction*.

Chapitre 4

Bilan

Le bilan de ce stage est pour moi très positif. Cela m'a permis d'aborder certaines technologies et certains principes que je n'ai pas étudié lors de ma formation. Cela m'a permi de me rendre compte de l'importance d'une analyse détaillée pour éviter de faire des erreurs qui peuvent amener à tout revoir.

4.1 Ce qui fonctionne

cœur du système Chaque utlilisateur est actuellement en mesure d'intervenir sur la base de données suivant son profil.

- Il peut, s'il en a le droit, créer un nouvel ouvrage ou une nouvelle revue ;
- il peut ajouter ou retirer n'importe quel champs ou groupe de champs, implémenté dans l'application ;
- il dispose d'un moteur de recherche assez efficace pour retrouver un livre ou une revue en saisissant un nom d'auteur, un ou plusieurs mots d'un titre, l'ISBN de l'ouvrage, son type de notification et sa référence ;
- il dispose également d'une barre de navigation qu'il lui permet aisément de revenir à une étape précédente.

Interaction avec le site Web Un utilisateur avec des droits sur le site Internet peut :

- ajouter un ouvrage sur le site à l'endroit où il désire ;
- mettre à jour un ouvrage qui est en ligne.
- retirer un ouvrage du site ;

Possibilité de répondre aux requêtes distantes. L'application répond aux requêtes distantes sur la demande d'ouvrages :

- par auteur ;
- par editeur ;
- par auteur et editeur ;
- par mot(s) de titre ;
- par ISBN.

Soit elle renvoie une chaîne XML au format ONIX contenant les ouvrages complets, soit une liste d'objet définis par :

- un titre ;
- des auteurs ;
- l'url du livre.

4.2 Ce qu'il reste à améliorer

Gestion des droits (notion de groupe) L'amélioration de la gestion des droits soulevée lors du projet précédent n'a pas avancé car elle ne faisait pas partie des priorités fixées par mon maître de stage. L'idée étant de regrouper les utilisateurs suivant leus fonctions qu'ils ont en commun.

La gestion des droits n'est pas totalement opérationnelle car il apparaît qu'un utilisateur n'ayant pas de droit en écriture puisse quand même modifier certain champs, ce qui nécessiterait encore quelques recherches, le principal étant le bon fonctionnement du *noyau* et des Web Service en priorité.

Gestion de l'application Il serait également utile de pouvoir modifier tous les fichiers de configuration de manière intuitive avec des interfaces dédiées.

Conclusion

Nous avons abordé dans ce rapport les besoins que pouvait avoir une maison d'édition en matière d'échange de données bibliographiques et surtout la nécessité de centraliser ces données.

La reprise de l'analyse du précédent projet a permi de répondre plus précisémment aux besoins des utilisateurs. On a démontré tous les avantages que pouvait apporter le langage XML dans une telle application.

Cette expérience m'a permi de réaliser à quel point une bonne analyse était importante. Même s'il y a déjà un existant, il peut être intéressant de reprendre l'analyse du départ et gagner peut être ainsi, beaucoup de temps et d'efficacité.

J'ai également appris au cours de ce stage à quel point il est impotant de se concevoir un système de journal d'erreur (log) efficace et clair, ainsi que de créer ses propres exceptions. On peut ainsi diminuer considérablement le temps de déboggage.

Ce stage à été pour moi très riche d'apprentissage dans le domaine des technologies récentes (J2EE, Axis, Strut, JSPX...) et plus particulièrment dans celui des Web Services (Axis, SOAP, WSDL).

Les Presses universitaires de Caen disposent maintenant des fondations de leur système centralisé sur lequelle pourront s'ajouter de nombreuses fonctionnalités (suivi de gestion, suivi de fabrication...).