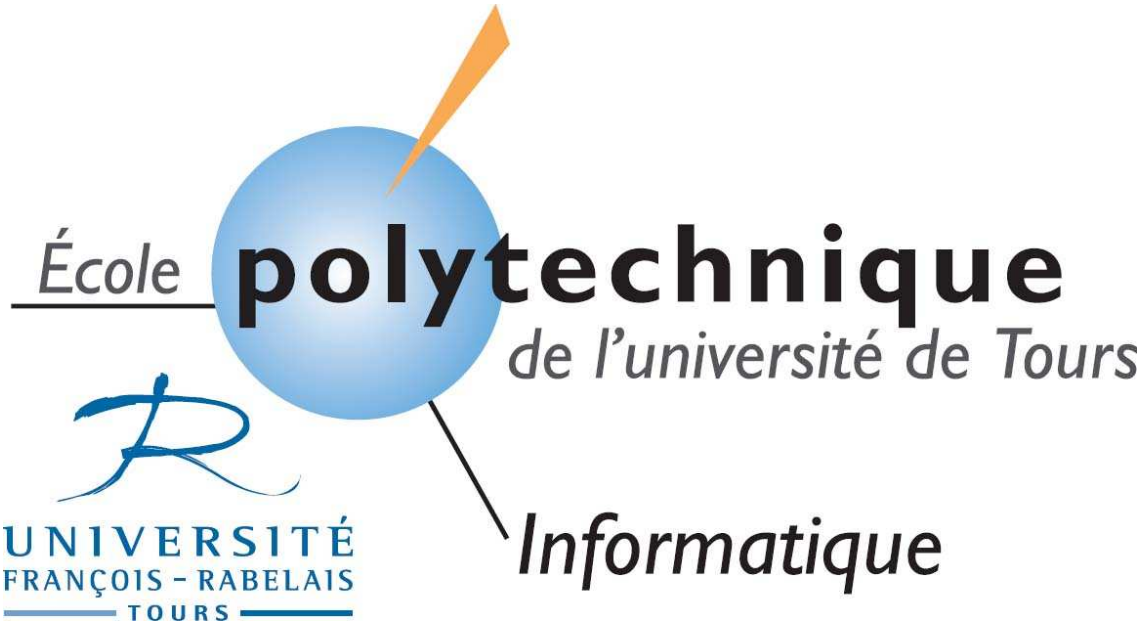
ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE TOURS
DÉPARTEMENT INFORMATIQUE
64, Avenue Jean Portalis
37200 TOURS



Rapport de Stage
Développement sur l'ERP OfbizNéogia

Encadrants :

Olivier HEINTZ
Néréide
3 bis, les Isles - 37270 Véretz

Etudiant :

Simon BAUDRY
EPU-DI 2ème année
Année - 2005-2006

Remerciements

Mes remerciements vont tout d'abord à Messieur Peter Goron et Olivier Heintz pour leur patience, la clarté et la précision de leurs explications et surtour pour leur bonne humeur et leur disponibilité.
Je souhaite également remercier l'enssemble de l'équipe Néréide pour l'aide et avant tout pour la convivialité qui a régné dans la société.
Je souhaite enfin remercier les stagaires présents dans l'entreprise en même temps que moi, Cédric Vallée, Mickaël Hardouin et Cécile Bourreau pour la bonne entente et les éclaircicements.

Table des matières

Introduction

Mon stage de 2ème année de l'Ecole Polytechnique Universitaire de Tours, s'est déroulé au sein de la société Néréide, SSLL (Solutions et Services en Logiciels Libres), spécialisée dans le domaine de l'intégration de PGI Open Source à destination des PME.
Actuellement, de plus en plus de développeurs et d'entreprises utilisent des logiciels Open Source comme MySQL, Eclipse, PHP ou JBoss et le système d'éditeur de logiciel propriétaire entame un déclin dû à un coût de licence prohibitif. Les entreprises qui investissent dans des solutions logicielles propriétaires pour améliorer leur système d'information sont généralement des grosses sociétés qui ne font encore que peu confiance aux logiciels libre malgré le peu d'arguments expliquant le coût de licence de ces logiciels. Seule, les offres de service associées trouvent des justifications.
En effet le principal avantage du logiciel libre est la réduction des coût tout en limitant les riques car celà grantit le respect de standard et la mobilisation des communautés très réactive aux nouveaux besoins.
La première partie de ce rapport portera sur le fonctionnement du PGI OfbizNéogia ce qui a constitué la première étape de mon travail dans l'entreprise puis nous verrons les développements éffectués et notament sur le projet Lola.

Chapitre 1

Présentation générale

1.1 L'entreprise

« La collaboration et les bonnes pratiques en action. »

Néréide est une jeune SSLL, fondée en 2004 et spécialisée dans l'intégration de l'ERP Open Source Ofbiz-Néogia auprès des PME. Il s'agit d'une SARL à capital variable. Son siège social se situe à une dizaine de kilomètres de Tours :

Société Néréide
3 bis, Les Isles
37270 VERETZ

En complémentarité avec l'intégration d'OFBiz-Néogia, cette société propose une gamme complète de services :

- développement d'applications spécifiques : réaliser ou accompagner le développement de fonctions complémentaires à OFBiz-Néogia et spécifiques au client ;
- administration de systèmes : mettre en place un élément système, un réseau, intégrer et migrer des composants systèmes, administration de système existant ... ;
- maintenance et support applicatif (TMA : Tierce Maintenance Applicative) : prestations de support et de maintenance corrective et / ou évolutive pour toutes les applications développées à partir de l'architecture OFBiz-Néogia ;
- gestion du système d'information (infogérance) : gestion globale du système d'information du client, prestation effectuée en trois phases : inventaire détaillé, transition et transformation. De part sa participation au réseau Libre-Entreprise son offre de service peut être étendue à toutes les compétences des membres de ce réseau.

L'équipe de Néréide est actuellement composée de 8 personnes. L'équipe technique est constituée de personnes expertes dans la mise en oeuvre de PGI Progiciel de Gestion Intégré, (essentiellement Baan, SAP, Oracle) issues de grands cabinets de conseils internationaux.
Durant le stage étaient également présents trois autres stagaires :

- Cedric Vallée, Département informatique de Polytech' Tours ;
- Mickaël Hardouin, Master informatique ;
- Cécile Bourreau, Master informatique ;

1.2 Le réseau Libre-Entreprise

Le réseau Libre-entreprise est un rassemblement d'entreprise toutes fortement impliquées dans le domaine du logiciel libre. Les membres du réseau ont en commun des valeurs et un mode de fonctionnement basé sur :

- le partage des connaissances ;
- la capitalisation des expériences clients et des réalisations ;
- le respect de tous les acteurs d'un projet ;
- la qualité des prestations.

L'appartenance d'une entreprise au réseau fait l'objet d'une évaluation par les autres membres sur le respect de ces valeurs qui forment la base du réseau. Un compte rendu mensuel permet d'avoir une idée de la situation précise de chaque entreprise, il porte sur leurs activités, leurs finances ... De nombreux documents sont partagés afin d'aider les membres du réseau dans leur démarches :

- documents sur la création de l'entreprise ;
- documents techniques ;
- modèles de documents ;

Un ensemble d'outils de travail collaboratif est mis à la disposition des membres du réseau pour simplifier la communication et le partage des connaissances :

- calendrier ;
- listes de diffusion partagées ;
- serveur de messagerie instantanée (Jabber) ;
- le laboratoire Libre-entreprise : une plate-forme d'hébergement de projets informatiques tel que le célèbre SourceForge. Elle offre les mêmes services, à savoir, site web, espace ftp, accès cvs, mailing-lists, etc. ;
- Planet Libre-Entreprise : c'est un aggrégateur de contenu qui permet de suivre l'activité des membres du réseau.

Chapitre 2

L'ERP OfbizNéogia

2.1 Les ERP

2.1.1 Définition

Un PGI progiciel de gestion intégré (en anglais Enterprise Resource Planning ou ERP) est un « logiciel qui permet de gérer l'ensemble des processus d'une entreprise en intégrant l'ensemble des fonctions de cette dernière comme la gestion des ressources humaines, la gestion comptable et financière, l'aide à la décision, mais aussi la vente, la distribution, l'approvisionnement, le commerce électronique »(Définition du grand dictionnaire terminologique de l'Office québécois de la langue française (OLF))

Le principe fondateur d'un ERP est de construire une application paie, (comptabilité, gestion de stocks) de manière modulaire tout en partageant une base de données unifiée. Cela crée une différence importante avec la situation pré-existante car les différentes fonctions de l'entreprise étaient gérées par une multitude d'applications dédiées souvent hétérogènes. Ainsi, les Achats, la Comptabilité, la Gestion des Stocks, les Ressources Humaines, la Gestion Commerciale,... sont maintenant totalement interconnectés. Avec l'arrivée de l'ERP, les données sont désormais standardisées et partagées entre les différents modules, ce qui élimine les saisies multiples et évite l'ambiguïté des données multiples de même nature (ex : «Clermont-Fd », « Clermont Ferrand », « Clermont-Ferrand », ...).

Ceci permet un accroissement considérable de la fiabilité des informations puisque la source des données est unique, d'où une réduction des délais et des coûts de traitements.
L'autre principe qui caractérise un ERP est l'usage systématique de ce qu'on appelle un moteur de workflow, et qui permet, lorsqu'une donnée est entrée dans le système d'information, de la propager dans tous les modules du système qui en ont besoin, selon une programmation prédéfinie.

Ainsi, on peut parler d'ERP lorsqu'on est en présence d'un système d'information composé de plusieurs applications partageant une seule et même base de données, par le biais d'un système automatisé prédéfini éventuellement paramétrable (un moteur de workflow).

Plus qu'un simple logiciel, un ERP est un véritable projet demandant une intégration totale d'un outil logiciel au sein d'une organisation et d'une structure spécifique, et donc des coûts importants d'ingénierie. D'autre part sa mise en place dans l'entreprise entraîne des modifications importantes des habitudes de travail d'une grande partie des employés.

2.1.2 Avantages

Comparés à des applications sur mesure, les ERP / PGI présentent plusieurs avantages :

- optimisation des processus de gestion (flux économiques et financiers) ;
- cohérence et homogénéité des informations ;
- intégrité et unicité du système d'information ;
- partage du même système d'information facilitant la communication interne et externe ;
- globalisation de la formation (même logique, même ergonomie) ;
- maîtrise des coûts et des délais de mise en oeuvre et de déploiement.

Il est important de remarquer que la mise en place d'un ERP dans une entreprise est souvent le déclencheur d'une réorganisation et rationalisation de l'ensemble des tâches et processus de l'entreprise.

2.1.3 Inconvénients

Les ERP / PGI ne sont cependant pas exempts d'inconvénients :

- coût élevé ;
- périmètre fonctionnel souvent plus large que les besoins de l'organisation ou de l'entreprise (le progiciel est parfois sous-utilisé) ;
- lourdeur et rigidité de mise en oeuvre ;
- difficultés d'appropriation par le personnel de l'entreprise ;
- nécessité d'une bonne connaissance des processus de l'entreprise (par exemple, une commande d'achat et une commande de vente nécessitent deux processus différents : il est important de savoir pourquoi, de savoir décrire les points communs et les différences entre ces deux processus de façon à bien les paramétrer) ;
- nécessité d'adapter parfois certains processus de l'organisation ou de l'entreprise au progiciel ;
- nécessité d'une maintenance continue.

2.1.4 Les ERP libres

Le secteur des ERP a depuis quelques années déjà subi un petit bouleversement : l'arrivée de logiciels libres (OFBiz Tiny ERP, ERP5, Compiere, ...) sur des terres où règnent en maîtres les logiciels propriétaires : SAP, BAAN, Oracle, ...
Le premier avantage des ERP Libre sur leurs alter-ego propriétaires est bien sûr l'absence de coût de licence ; coût qui peut souvent apparaître comme prohibitif pour les PME.
Un autre atout important est la possibilité d'adapter et de faire évoluer soi-même le progiciel sans dépendre du bon vouloir de la société éditrice. En outre, le logiciel libre mobilise souvent des communautés, qui le font évoluer au gré des nouveaux besoins, et qui peuvent répondre rapidement à des demandes précises.
De plus comme tout logiciel libre, les ERP libre donne la garantie de travailler sur des standards ouverts et donc inter-opérables, avantages stratégiques pour beaucoup d'entreprises.

En parallèle avec l'augmentation de l'utilisation des ERP en France 48 % des PME françaises sont équipées, et en janvier 2005, 9 % des PME françaises envisageaient d'acquérir et de mettre en place un nouvel ERP dans l'année (Atelier groupe BNP Paribas), l'intérêt porté par les entreprises sur le logiciel libre progresse. Ainsi, 58 % des entreprises envisageraient de passer de leur ERP propriétaire actuel à un ERP libre en 2004 (ERP2004 INFOWORLD).
La présence du logiciel libre sur le marché des ERP n'est donc plus marginal et les ERP Open Source prennent leurs places dans ce secteur.

2.2 Ofbiz

2.2.1 Présentation

Ofbiz est un logiciel de gestion de production communément appelé PGI. Ce projet a été initié en mai 2001 par deux américains David E. Jones et A. Zeneski. Leur souhait étaient de créer des outils et une application pour le commerce. Dès le début les outils et l'architecture ont été développé afin d'une part, de faciliter le développement de fonctionnalité et d'autre part, de réduire le temps de maintenance du code existant.

Une des caractéristiques fortes de ce PGI est son appartenance au monde des logiciels libres. En effet, les deux concepteurs ont appliqué une licence « MIT Open Source License » qui implique le libre accès aux sources mais également le droit de les modifier, de les utiliser et même de les vendre. La seule contrainte de cette licence est de respecter le copyright des sources.

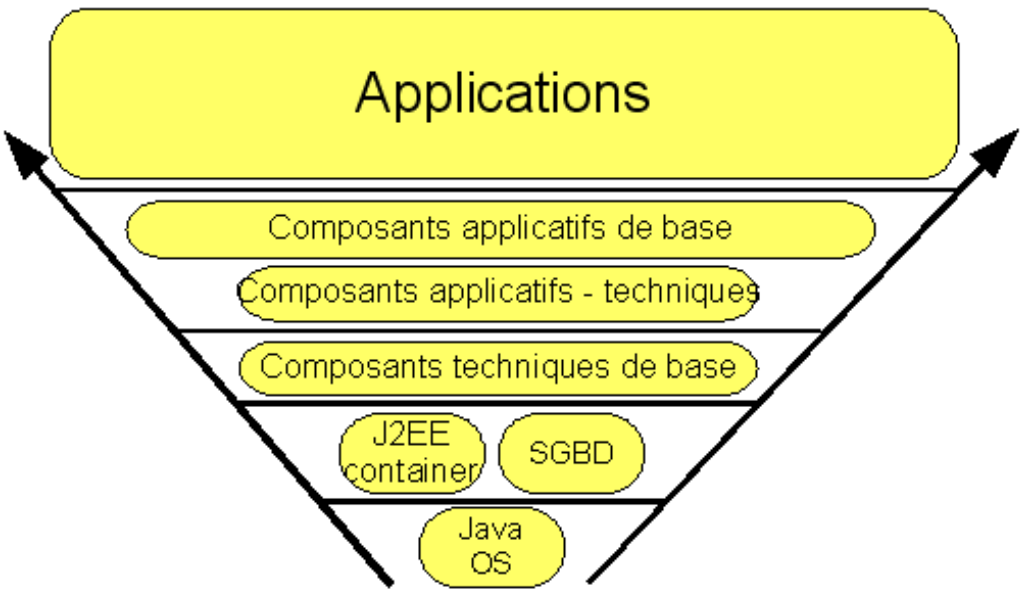
Aujourd'hui, le projet rassemble une quinzaine de développeurs dans le monde. Ofbiz est fortement orienté vers le e-commerce mais son architecture modulaire a permis d'intégrer des composants dédiés à l'ensemble des fonctions que l'on retrouve dans différents types d'entreprise (service, industrie, commerce...).

2.2.2 Architecture

Ofbiz est une application java client-serveur compatible avec la spécification J2EE définissant une architecture logicielle standard. On retrouve ainsi les trois éléments caractéristiques d'une architecture 3-tiers :

- les clients : ici des clients légers, typiquement des machines peu puissantes disposant d'un navigateur internet ;
- un serveur exécutant les différentes applications Ofbiz ;
- une ou plusieurs bases de données stockant le système d'information de l'entreprise.

OFBiz repose sur une architecture à plusieurs niveaux où chaques niveaux sont spécialisés dans un domaine.



Ils sont composé de plusieurs éléments spécialisés dans la réalisation de tâches bien précises.

2.2.3 Framework

Ofbiz est en premier lieu un "framework d'application d'entreprise" dans lequel chaque composant représente une brique logicielle pouvant être réutilisée pour construire des applications diverses. Ce framework repose sur trois composants essentiels sans lesquels une application standard ne pourrait pas fonctionner : l'Entity Engine, le Service Engine et le ControlServlet.

L'Entity Engine

L'Entity Engine est un composant Ofbiz qui se charge de gérer les transactions avec la base de données. Il est constitué d'une collection d'API qui implémente des objets java qui vont accéder aux données de la table via des méthodes. Ainsi le développeur n'accède pas aux données directement mais via une couche abstraite qui se charge d'accéder aux données et de sécuriser les transactions. Le code est ainsi indépendant du SGBD et le développeur n'a pas à connaître le langage SQL.

Ses principales caractéristiques sont :

- accès aux données via une interface unique, le "GenericDelegator".
- supporte l'accès transparent à plusieurs base de données.
- les entités sont définies dans de simples fichiers XML.
- tous les types java de base ont un équivalent en base de données.
- supporte les transactions distribuées.
- suppporte un mécanisme de trigger appelé "EECA" (Entity Event-Condition-Action) même si le SGBD sous-jacent n'implémente pas cette fonctionnalité.

Le Service Engine

Le Service Engine est l'équivalent de l'Entity Engine pour tous les traitements des composants Ofbiz. Les traitements sont appelés Services et peuvent être exécutés localement ou à distance. Le principal intérêt de ce composant est qu'il permet de lancer des services sans avoir besoin de connaître leur localisation et leur implémentation. C'est le ServiceDispatcher qui se charge alors de trouver l'implémentation du service et de son exécution. Un autre intérêt est la possibilité de rendre disponible tout service Ofbiz vers l'extérieur. Ainsi un composant peut appeler un service d'un autre composant.
Ils sont codés soit en XML Mini-Language qui est un outil d'ofbiz soit codés directement en java, ils peuvent alors appeler d'autres méthodes java ou soit codés dans un autre langage (comme python, perl, etc ...).
Les services sont définis par des fichiers xml. L'utilisation des services permets de garantir une uniformité dans la manipulation des données et d'engendrer des contrôles tel que le login ou la gestion des erreurs.

Le ControlServlet

Le ControlServlet est l'élément clé de la communication entre les utilisateurs et les applications web d'Ofbiz. Implémenté selon le modèle MVC (Modèle-Vue-Controleur), il gère la boucle d'évènements de l'interface graphique et les différents moteurs de rendu de l'application. Les réponses aux intéractions de l'utilisateur s'effectuent par l'intermédiaire d'évènements qui peuvent être implémentés sous la forme de services, de méthodes java, de scripts Beanshell ou Minilang.
Un moteur de rendu se charge de renvoyer à l'utilisateur une vue d'un document généré à partir des évènements précédents et/ou de données.
Les moteurs de rendu dont dispose Ofbiz :

- ***JSP*** : Les Java Server Pages permettent de générer des pages HTML dynamiquement.

- ***JPublish + FreeMarker*** : JPublish permet de construire une page HTML à partir de plusieurs fichiers traités par FreeMarker et d'appeler pour chacun d'entre-eux des scripts BeanShell à différents moments de la construction. FreeMarker est un moteur de template qui permet

de générer des documents dynamiquement.

- ***FOP + FreeMarker*** : FOP est un processeur XSL qui transforme un document XML traité avec FreeMarker en un document HTML ou PDF.

- ***JasperReport*** : JasperReport permet de réaliser très facilement des rapports à partir de multiples sources de données.

Voici l'ensemble des opérations effectuées suite à une intéraction avecl'utilisateur pour lui afficher une page à l'aide de JPusblish et FreeMarker :

1. L'utilisateur clique sur un lien hypertexte ou valide un formulaire. Le navigateur envoie alors une requête HTTP au serveur Ofbiz qui est interceptée par Tomcat et transmise au ControlServlet de l'application web correspondante.$ex : https : //127.0.0.1 : 8443/ordermgr/control/orderview?order_id = WS10000$
2. Le ControlServlet vérifie si l'URI demandée est définie par l'application. Le cas échéant, il appelle le ou les événements associés à cette URI. Dans le cas contraire, il renvoie une erreur au navigateur web de l'utilisateur (Erreur HTTP 404 : page non trouvée).
3. Si l'évènement généré doit appeler un service, il vérifie que les paramètres de la requête correspondent aux attributs du service.
4. Si l'évènement généré doit appeler un service, il convertit les paramètres de la requête sous forme textuelle en objets Java correspondant.
5. L'évènement appelle un service ou un gestionnaire d'évènements (méthode java statique).
6. Le service ou le gestionnaire d'évènements peuvent effectuer des actions sur le modèle de données.
7. L'EntityEngine convertit ces actions en requêtes SQL pour le serveur de base de données.
8. Le service ou le gestionnaire d'évènement renvoie le résultat de leur action.
9. L'évènement transmet ce résultat au ControlServlet.
10. À partir du résultat de l'évènement, le ControlServlet sélectionne la vue à afficher et appelle le moteur de rendu adéquat.
11. À partir de la définition d'une vue, le moteur de rendu construit les différents sous-éléments de cette dernière.
12. Pour chaque sous-élément, il peut appeler des scripts BeanShell qui récupèrent et mettent en forme les données à afficher.
13. Pour chaque sous-élément, il appelle le moteur de template qui se charge de générer le code HTML correspondant.
14. Le moteur de rendu assemble les différents sous-éléments pour former une page web complète.
15. Le controlServlet transmet la page générée au navigateur web de l'utilisateur.

2.2.4 La couche métier

La couche métier d'OFBiz est composé de modules. Chaques modules correspondent à une application métier.

Organisation d'un module

Un module est un programme fonctionnel s'appuyant sur les couches techniques du PGI. Le module contient toutes les informations nécessaires à son bon fonctionnement : description des entités, source java, description des services, fichier de définition d'écran, etc ...
Toutes ces informations sont organisées dans une arborescence propre au module. Cette arborescence la même structure quelque soit le module et sera détaillée au chapitre suivant.
Les modules peuvent inter-agir entre eux via des mécanismes simples (service), permettant de bien spécialiser les fonctionnalités de chacun.

Architecture d'un module

Les modules sont tous structurés de la même façon. Ci-dessous suit une présentation de l'architecture d'un module d'OFBiz :

```
module
|
+- config
|
+- data
|
+- entitydef
|
+- lib
|
+- servicedef
|
+- src
|
+- webapp
| |
| '-module
| |
| +-error
| |
| +-sous-module
| |
| +-templates
| |
| +-WEB-INF
| | |
| | +-actions
| | |
| | +-pagedefs
| | |
| | |-controller.xml
| | |
| | |-jpublish.xml
| | |
| | '-web.xml
| |
| '-login.ftl
|
'- OFBiz-components.xml
```

Description des fichiers et répertoires

module :

config : ce répertoire contient les fichiers d'internationalisation et les fichiers « properties » de configuration.

data : ce répertoire contient des fichiers xml de données utilisateurs relatif au module. Ces données sont d'ordre d'obligation ou de démonstration

entitydef : ce répertoire contient deux fichiers qui décrivent les entités qu'utilisent ce module. Ces fichiers sont directement utilisés pour la description et la construction des tables de la base de données.

servidef : ce répertoire contient un fichier, services.xml qui associe des fonctions java, ou d'autres langages, présentes dans les sources du module à un nom de service.

src : ce répertoire contient les sources des fichiers java relatif au module. C'est dans ces fichiers que sont définies les fonctions appelées via le fichier services.xml.

webapp : ce répertoire contient une arborescence de fichiers qui sont utilisés pour la génération des écrans utilisateurs.

OFBiz-components.xml : ce fichier décrit le module pour son intégration dans OFBiz.

module/webapp/module :

error : ce répertoire contient les messages à afficher suivant l'erreur générée par le serveur d'applications.

sous-module : un module est organisé en sous-parties, suivant l'application faite dans le module avec un objectif de description. Ce répertoire contient les fichiers de structure des forms et les fichiers de

structure d'écrans utilisateurs (fichiers ftl).

templates : ce répertoire contient les fichiers de structure de base comme l'écran d'accueil du module ou encore les écrans de recherche.

WEB-INF : ce répertoire contient une arborescence contenant les fichiers effectuant les traitements nécessaires sur les informations avant leur affichage. l

ogin.ftl : ce fichier décrit la structure de l'écran de connexion au module.

module/webapp/module/WEB-INF :

actions : ce répertoire contient des fichiers source java interprétés : des fichiers beanShell. Ces derniers servent à remplir le contexte d'informations, nécessaire à l'affichage des informations dans les écrans

utilisateurs. Ces informations peuvent provenir d'un appel à la base de données ou d'un traitement quelconque sur des informations déjà présentes dans le contexte.

pagedefs : ce répertoire contient la définition d'un écran utilisateur. Ceci comprend les informations initiales et/ou à rajouter au contexte pour l'utilisation de cette page ainsi que diverses définitions des fichiers de structure et des fichiers BeanShell à appeler pour la génération de l'écran.

controller.xml : ce fichier met en relation les urls envoyées par le client et la page à renvoyer. Cette page est le résultat d'un enchaînement d'opérations qui sont indiquées dans ce fichier.

jpublish.xml : fichier de configuration pour la fusion du contexte avec les fichiers de strucutre ftl.

web.xml : configuration du serveur web.

Deux fichiers sont extrêmement importants dans un module : le fichier controller.xml et le fichier services.xml.

Le premier : controller.xml sert à OFBiz pour savoir quel service et action puis quelle vue doit être appelés suivant une requête effectuée.

Le second : services.xml sert à OFBiz pour mettre en relation l'appel à un service et la fonction associée se trouvant dans les sources du module

2.2.5 Les modules

Content

Le module Content permet d'assurer la gestion de contenu (CMS). Ses entités sont utilisées pour enregistrer et manipuler les contenus généraux et les bases de connaissance. Ces entités incluent de nombreux concepts tels que : la séparation de l'information et de l'organisation des données qui peut être utilisé dans beaucoup de structures de données comme des arbres, listes ou des Maps d'objets. Une fois ces structures créées, des outils évolués de recherche d'information sont utilisés pour automatiser la création de nouvelles structures et permettre à l'entreprise de gérer les documents.

Accounting

Les entités de Comptabilité sont organisées sur des principes généralement admis comme la comptabilité à double entrée, un registre général avec des comptes hiérarchisés... Elles sont structurées pour que l'on puisse gérer la comptabilité de plusieurs organisations.

Party

Le module Party permet d'assurer la gestion de la relation client (CRM). Un Party peut représenter soit une personne physique soit un groupe (un groupe pouvant être une entreprise, un fournisseur ou un ensemble de personnes). La notion de groupe permet de modéliser des hiérarchies, des groupes de sécurité. Cette application est généralement utilisée pour gérer les informations sur le personnel de l'entreprise, sur les relations avec ses clients et ses fournisseurs, etc. À chaque contact, on peut associer de nombreuses informations telles que des adresses, des numéros de téléphones, des rôles, et par un mécanisme d'extensions, des données supplémentaires.

Product

Les entités de Product contiennent les informations générales sur les produits vendables, achetables d'une entreprise. Les produits peuvent être des articles (matières premières, produits finis ...), des services, ... Les produits peuvent être organisés en catégories et en catalogue (notion de promotions, canaux de ventes...). Ils peuvent être associés à une multitude de prix selon la devise, le fournisseur, les dates, la quantité achetée, etc.

Facility

Un « Facility » est un bâtiment ou un emplacement physique tel que les stocks, les magasins, les docks, les bureaux,... En général un « Facility » aura un contact associé : une adresse, un numéro de téléphone, ... Les bâtiments peuvent être regroupés en groupe de bâtiments, eux-mêmes pouvant faire partie de groupes de bâtiments. Ces groupes sont, par exemple, des chaînes de magasins, régions, départements. Des personnes ou groupes de personnes peuvent aussi être associés à des bâtiments pour définir où une personne travaille, qui gère le bâtiment, etc. Ce module permet de gérer les stocks d'une entreprise, il connaît ainsi pour un produit ses lieux de stockages, les quantités stockées et les indices de gestion de stock : seuils d'alerte, quantité économique...

Order

Le module « Order » permet de gérer tous les processus autour d'une commande d'achat ou de vente. Un ordre se compose d'une en-tête de commande et de lignes de commandes qui décrivent les détails de l'ordre et des ajustements tarifaires. Ces ajustements correspondent aux promotions, aux taxes et aux frais de ports appliqués à l'ordre. Toutes les étapes d'une commande sont gérées du devis, à la facturation en passant par la réception de la commande, la gestion du retour de marchandise, ...

Shipment

« Shipment » gère l'ensemble des échanges de produits avec l'extérieur, autrement dit les réceptions et les expéditions ainsi que les entrées et sorties de stock. On peut ainsi connaître pour un produit et un « Shipment » la quantité du produit expédiée ou reçue. Shipment fait aussi le lien avec les services des transporteurs pour le suivi des colis et des livraisons.

2.2.6 Domaine d'application

Le framework d'Ofbiz fournit par défaut pratiquement toutes les applications nécessaires à quasiment tous les domaines de l'entreprise. Celà constitue l'un de ses principaux avantages par rapport à d'autres framework du même type. Ces applications sont fournies de tel sorte qu'elles sont prêtes à l'emploi, il ne reste à l'utilisateur qu'à saisir les données spécifiques à l'entreprise.
Il existe également un autre type d'applications dan Ofbiz, les applications métiers. Elles sont spécifiques à un domaine d'activité.

2.3 OfbizNéogia

2.3.1 Naissance du projet

Bien qu'OFBiz soit totalement écrit en Java, la modélisation utilisée est un modèle entité-relation. Ainsi, les entités de base de données ne sont pas traduites en objet, on accède donc directement aux tûples de la base de donnée. La puissance d'un langage objet est alors totalement sous-utilisé, la plupart des méthodes est statique, on perd de plus le haut niveau d'abstraction offert par le langage objet.
Le modèle entité-relation qui s'attache à la modélisation des données s'avère peu adapté à la réalisation des composants métiers où la modélisation des traitement est à privilégier. Néréide a donc initié en mai 2004, le projet Néogia publié sous licence GPL. Cette licence GPL moins permissive que la la licence MIT assure que le code ne sera jamais commercialisé. Ce projet a pour but de fournir des outils permettant de créer des composants OFBiz grâce à une modélisation UML.
Les développements à venir sur OFBiz ne concernent que le dernier niveau du PGI : la création de la couche métier. Or, les faiblesses indiquées au chapitre précédent ne nous poussent guère à lancer un développement massif pour combler les manques fonctionnels du PGI. C'est pour ces raisons que commença un nouveau projet nommé Neogia.

Que doit apporter Neogia afin de faciliter le développement de la couche métier :

- une sur-couche objet pour appliquer un développement objet et ignorer la structure de la base de données.
- remplir automatiquement le plus de fichiers de configuration possibles dont les informations sont répétitives, pour les écrans standard.

2.3.2 Arborescence des fichiers

Neogia possède une arborescence de fichiers totalement séparée de celle d'OFBiz :

```
neogia
|
+- components
| |
| +- module1
| |
| +- module2
| | |
| | +-dist
| | |
| | +-target
| | |
| | +-src
| | |
| | +-build.xml
| | |
| | +-project.xml
| | |
| | +-project.properties
| | |
| | `- maven.xml
|
+- doc
|
+- generators
| |
| +-src
| |
| +-target
| |
| +- neogiaproject.xml
| |
| +- project.xml
| |
| +- project.properties
| |
| `- maven.xml
|
+- neogiaproject.xml
|
+- project.xml
|
+- neogiaOFBiz.patch
|
+- maven.xml
|
`- build.xml
```

Description des fichiers et répertoires

- Neogia : répertoire racine du projet
  - components : ce répertoire contient tous les modules en développement qui seront intégrés à OFBiz
    - module
      - dist : ce répertoire contient le module développé de Neogia qui est opérationnel dans OFBiz (fichiers générés + développements spécifiques)
      - src : ce répertoire contient le diagramme UML décrivant le module ainsi que quelques informations non générables comme des labels pour l'internationalisation du module
      - target : ce répertoire contient les fichiers générés à partir du diagramme UML, une fois la génération finie.
      - project.xml et project.properties : ce sont deux fichiers de configuration utilisés par LutinGenerator pour la génération
      - maven.xml : fichier de configuration pour le lancement de la génération via maven
      - build.xml : fichier d'instruction pour ant afin de déplacer les fichiers voulus dans l'arborescence d'OFBiz, utilisé seulement par les développeurs du module
  - generators
    - src : répertoire contenant les sources des générateurs
    - target : répertoire contenant les générateurs compilés
    - neogiaproject.xml, project.xml et project.properties : ce sont trois fichiers de configuration utilisés par LutinGenerator pour préparer les générateurs
    - maven.xml : fichier de configuration utilisé par maven pour compiler les générateurs
  - build.xml : ce fichier d'instruction ant déplace le contenu des répertoires components/*/dist vers l'arborescence OFBiz pour intégrer les modules Neogia dans le PGI.
  - neogiaOFBiz.patch : patch à appliquer à la racine d'OFBiz pour préparer le PGI à l'intégration des modules développés via Neogia
  - project.xml et project.properties : ce sont deux fichiers de configuration utilisés par LutinGenerator pour la description du projet
  - maven.xml : fichier de configuration pour la configuration du projet via maven.

2.3.3 Changement dans le mode développement

Neogia utilise un projet nommé LutinGenerator, développé par l'entreprise Code Lutin, qui est un « préparateur » à la génération. LutinGenerator fonctionne en deux temps :

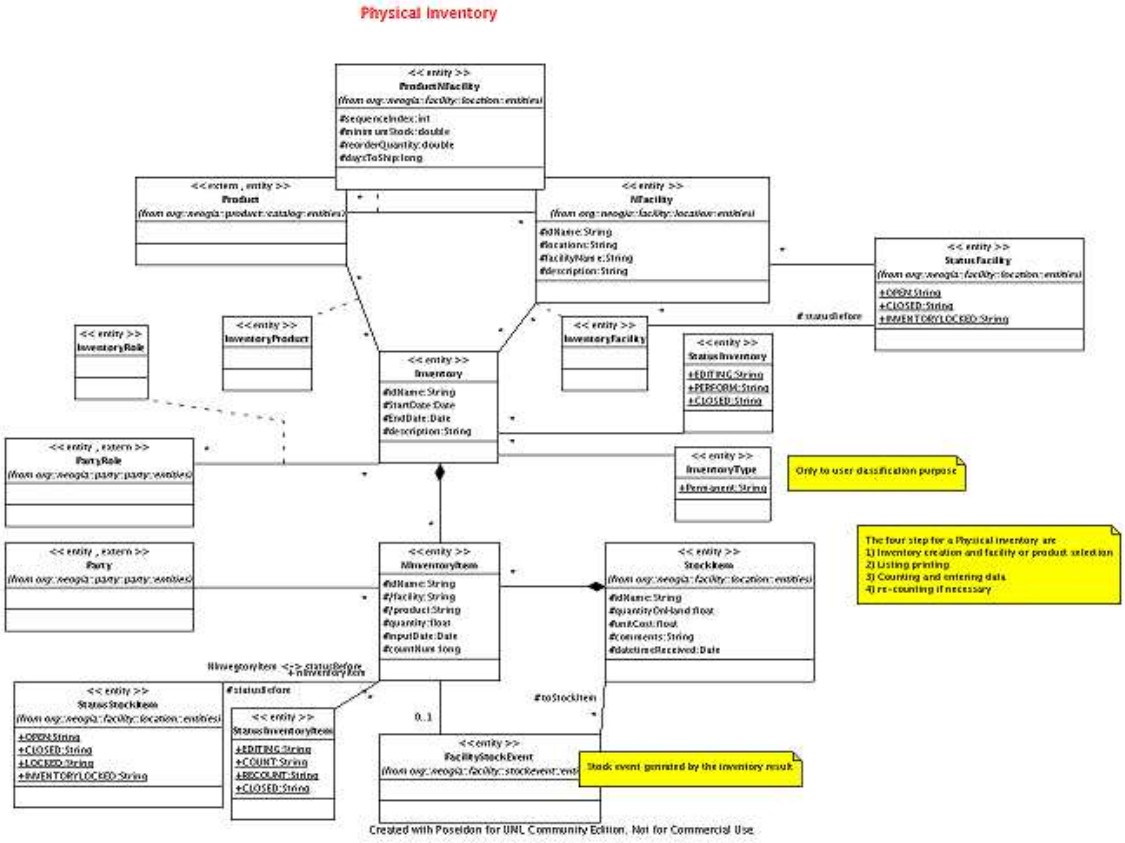
1er temps : LutinGenerator lit des fichiers scripts, typés sur java, et construit une librairie avec les informations contenues dans ces fichiers.

2ème temps : LutinGenerator lit un diagramme de classe UML et formate le flux récupéré. Ensuite le formatage est appliqué à la librairie qui génère une structure de fichiers sources définie.

Quels sont les intérêts de LutinGenerator :

- La génération se fait à partir d'un diagramme de classe UML, ce qui apporte une conception du logiciel de haut niveau.
- La génération n'est pas figée à un langage de programmation comme beaucoup de logiciels de création de schéma UML. Les générateurs sont écrits suivant ce que l'on désire générer, avant la première génération du diagramme.

Exemple de diagramme de classe utilisé :



La génération de code à partir de diagrammes UML permet de créer :

- fichiers de définition des services ;
- fichiers de définition des entités ;
- fichiers Java permettant l'abstraction Objet<->Entité ;
- fichiers d'interfaces graphiques par défaut pour les objets modélisés ;
- fichiers de formulaires de recherche ;
- fichiers d'internationalisation.

On peut aussi remarquer que la génération, de par son caractère systématique, permet d'amener une certaine homogénéisation des procédures de codage, homogénéisation malheureusement absente sur OFBiz.
De plus, l'utilisation préalable d'une modélisation UML au codage « direct » offre la possibilité de fixer clairement le fonctionnement du module et permet aux nouveaux développeurs d'avoir un point de départ clair et précis pour mieux comprendre l'implémentation adoptée.

2.3.4 Les apports de Néogia

Neogia est une sur-couche d'OFBiz dont le développement s'effectue en parallèle du développement de ce dernier. Il a pour objectif de combler les faiblesses d'OFBiz pour le développement de la couche fonctionnelle du PGI et d'en accélérer son évolution.

Neogia est un ensemble de composants complémentaires à la plate-forme d'application d'entreprise OFBiz.
Ces composants sont de 3 types :

- Des composants fonctionnels soient en tant que composant nouveau, soient remplacant un composant OFBiz existant.

  manufacturing : remplace le composant OFBiz existant, c'est une refactorisation de celui-ci, avec la définition d'un modéle UML propre et une ré-écriture compléte du code.

  facility : remplace le composant OFBiz existant pour toute la gestion des stocks, il n'inclut pas la gestion des expéditions qui reste réalisée par OFBiz. Il fournit une gestion des inventaires physiques complète. Ce composant est apparu suite à une refactorisation complète du modéle de données réalisé avec UML permettant de gérer les stocks actuels et planifiés.

  accounting : remplace le sous-composant OFBiz existant, pour la gestion comptable et analytique. La modélisation UML est entièrement nouvelle. La gestion des paiements reste réalisée par OFBiz

  servicemgnt : nouveau composant permettant de gérer des activités de service ou de projet. Le composant étant nouveau, son modéle UML est également nouveau.
- Des composants permettant de se connecter aux composants OFBiz existants. Les diagrammes UML reprennent les éléments de OFBiz.

  common : utilisé pour les liaisons avec les entity enum et status

  content : utilisé pour unifier certaines règles de développement et pour la gestion des champs en multi-langue.

  order : utilisé pour accéder aux objets commandes et ligne de commandes

  party : utilisé pour accéder aux objets acteurs, rôle, acteur-rôle et adresse, et aux objets communications

  product : utilisé pour accéder à l'objet article et pour la liaison entre OFBiz et le composant facility de Neogia

Un composant technique, permettant de générer la majeure partie du code OFBiz à partir des diagrammes de classe UML. Cela permet d'avoir des composants développés à partir d'une modélisation objet. La génération permet de généraliser les bonnes pratiques OFBiz et mets à disposition des développeurs les éléments nécessaires au développement objet. Les développements complémentaires sont réalisés dans des sur-charges objets et pas sur les éléments générés garantissant ainsi la possibilité de régénérer certains éléments lors de l'apparition de nouvelle bonne pratique.

2.3.5 En conclusion

Le développement via Neogia des modules fonctionnels d'OFBiz permet :

- une conception de haut niveau de chaque module.
- une génération de 70% à 80% des fichiers sources nécessaires au fonctionnement du module.
- une écriture des fichiers sources propre, structurée et directement en relation avec la conception.
- une sur-couche objet pour faciliter le reste du développement.

Chapitre 3

Travail effectué

3.1 Présentation des outils utilisés

3.1.1 Eclipse

Eclipse est un environnement de développement intégré (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en oeuvre n'importe quel langage de programmation. L'application est écrite en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.
Sa spécificité vient du fait de son architecture totalement développée autour de la notion de plug-in : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plug-in.

3.1.2 CVS

CVS, acronyme de Concurrent Versions System, est un logiciel libre (licence GPL) de gestion de versions. Puisqu'il aide les sources à converger vers la même destination, on dira que CVS fait la gestion concourante de versions ou de la gestion de versions concurrentes. Il est très utilisé dans le domaine du logiciel libre et dans les projets Ofbiz et OfbizNéogia. Il peut aussi bien fonctionner en mode ligne de commande, qu'à travers une interface graphique. Il se compose de modules clients et d'un ou plusieurs modules serveur pour les zones d'échanges.
Classiquement, un logiciel de gestion de versions va agir sur une arborescence de fichiers afin de conserver toutes les versions des fichiers, ainsi que les différences entre les fichiers.
Ce système permet par exemple de mutualiser un développement. Un groupe de développeurs autour d'un même développement se servira de l'outil pour stocker toute évolution du code source. Le système gère les mises à jour des sources par chaque développeur, conserve une trace de chaque changement. Ceux-ci sont, en bonne utilisation, chaque fois accompagnés d'un commentaire. Le système travaille par fusion de copies locale et distante, et non par écrasement de la version distante par la version locale. Ainsi, deux développeurs travaillant de concert sur un même source, les changements du premier à soumettre son travail ne seront pas perdus lorsque le second, qui a donc travaillé sur une version non encore modifiée par le premier, renvoie ses modifications.
Généralement, chaque version est incrémentée de 1 par rapport à la précédente. On l'appelle révision.
Dans le cadre de mon travail cet outil a été utilisé au travers du logiciel de développement Eclipse.

3.1.3 Modélisation UML : Poséidon

Parmi les nombreux outils de modélisation UML existant sur le marché, il en est un dont l'on parle de plus en plus, tant par ses fonctionalités que par sa gratuité : Poseidon for UML. Ce n'est pas un logiciel libre, mais la version commerciale du logiciel libre ArgoUML. Une version communautaire existe.
La société allemande Gentleware propose ce modélisateur depuis 2000, et a continué de faire évoluer le logiciel en fonction des différentes normes UML successives et spécifications de technologies annexes (XMI) devenues des standards.

3.1.4 Les constructeurs : Ant et Maven

En plus d'Eclipse et de Poséidon, on utilise le constructeur ant, qui est un projet open source de la fondation Apache écrit en Java qui vise le développement d'un logiciel d'automatisation des opérations répétitives tout au long du cycle de développement logiciel, à l'instar des logiciels Make.
Ant est principalement utilisé pour automatiser la construction de projets en langage Java, mais il peut être utilisé pour tout autre type d'automatisation dans n'importe quel langage.
Les concepteurs d'Ant ont indiqué vouloir faire un système ne souffrant pas des limitations de Make. Les scripts de compilation (appelés build.xml) sont écrits en XML. Il est possible d'étendre Ant afin de créer ses propres tâches à l'aide de module Java.
Parmi les tâches les plus courantes, citons : la compilation, la génération de pages HTML de document (Javadoc), la génération de rapports, l'exécutions d'outils annexes (checkstyle, findbugs etc), l'archivage sous forme distribuable (JAR etc.)
Quant à Maven il est semblable à l'outil Ant, mais fournit des moyens de configuration plus simples, basés sur le format XML. Maven est géré par l'organisation Apache Software Foundation. Précédemment Maven était une branche de l'organisation Jakarta Project.
Un élément clé et relativement spécifique de Maven est son aptitude à fonctionner en réseau. Une des motivations historique de cet outil est de fournir un moyen de synchroniser des projets indépendants : publication standardisée d'information, distribution automatique de modules jar. Ainsi en version de base, Maven peut dynamiquement télécharger du matériel sur des entrepôts logiciels connus. Il propose ansi la synchronisation transparente de modules nécessaires.
Cet outil est utilisé pour compiler le générateur de code.

3.2 Rêgles de développement

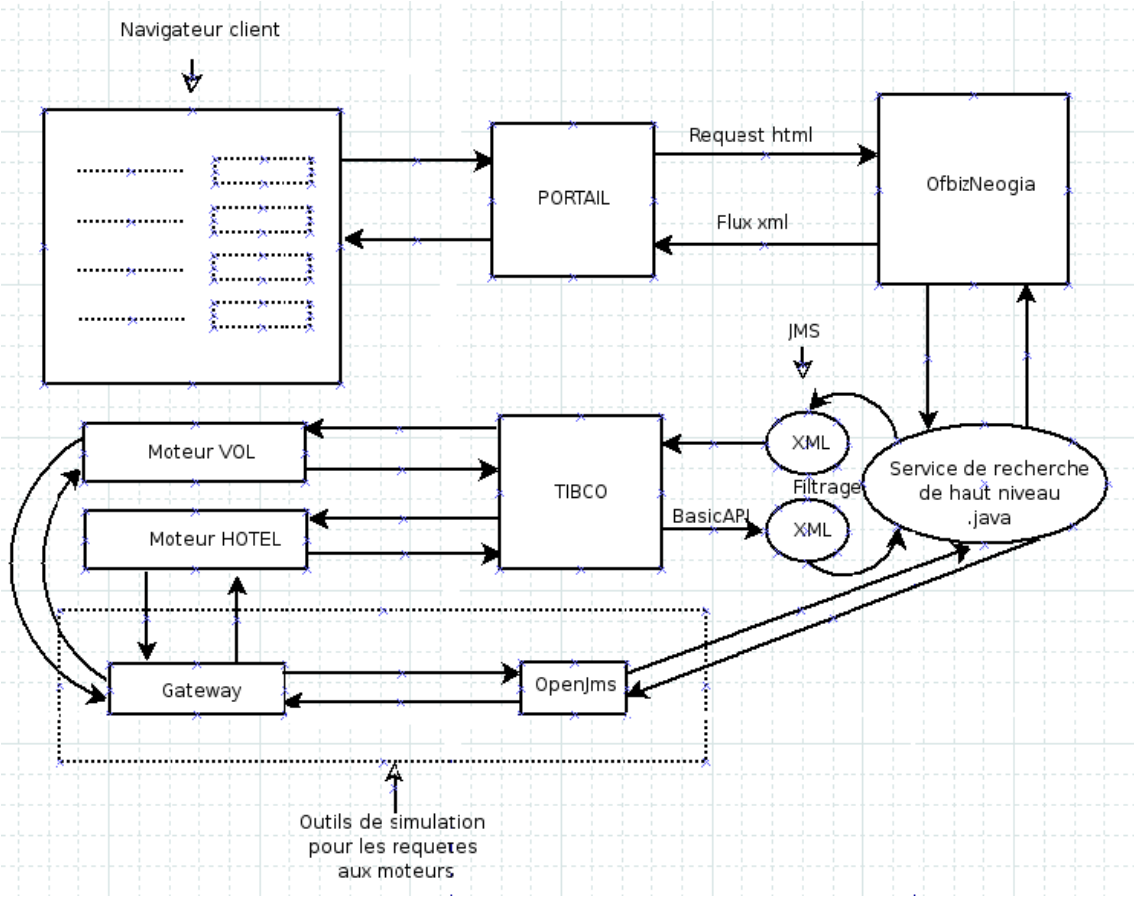
Le système de contrôle de versions utilisé sous Néogia est CVS. Le processus de développement avec OFBiz-Néogia est simple. Plusieurs branches sont utilisées, dont principalement la branche HEAD et la branche STABLE.
Les programmeurs travaillent avec la branche HEAD et valident directement leurs travaux sur cette dernière. Lorsque les fonctionnalités sont suffisamment mûres et solides, elles sont incorporées dans la branche STABLE. C'est cette branche STABLE qui est proposée aux clients.
Lorsque les programmeurs d'OFBiz-Néogia travaillent sur des composants propres à OFBiz, ce qui est relativement peu courant, nous envoyons nos modifications suivant la procédure d'OFBiz. Afin d'optimiser la communication entre les différent collaborateurs et ainsi la productivité, nous utilisons un repositoire Web. Les différentes tâches à accomplir y sont déposées et classées par type (bug, patch OFBiz, nouvelle foncionnalité) et module. Une priorité peut être associée à chaque tâche. Les status d'iune tâche sont les suivants

- ToBeTested : la tâche a été réalisé on fait appel à un collaborateur différent du réalisateur afin de valider le fonctionnement
- ToStable : la tâche a été validé et est prête à passer dans la branche STABLE
- ToJIRA : le développement réalisé doit être envoyé sous forme de patch à OFBiz

3.3 Le module Lola

3.3.1 Présentation

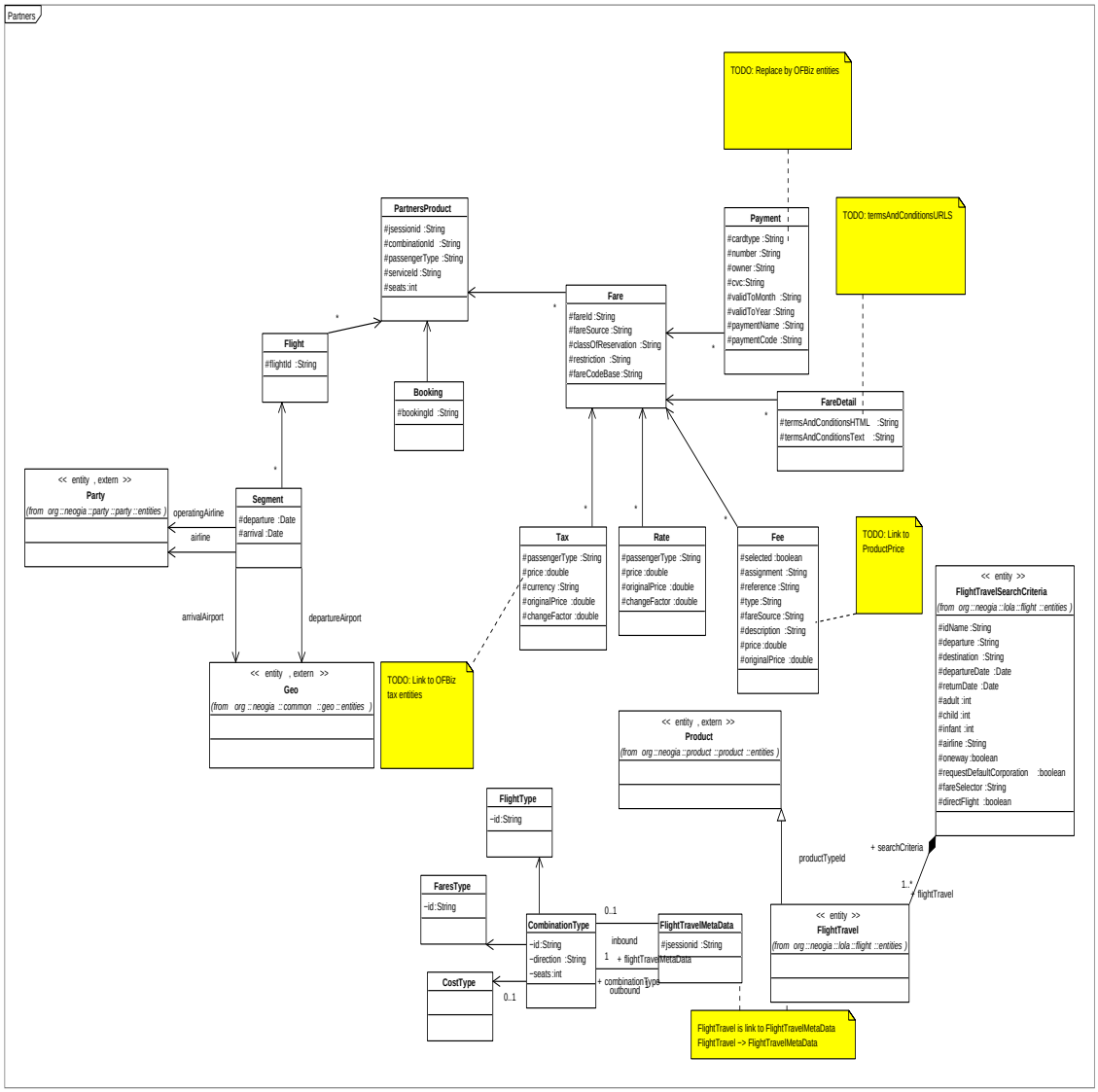
Au cours de mon stage j'ai travaillé sur diverses tâches et modules. Le travail des premières semaines à surtout servi à utiliser, à apprendre à maitriser les différents outils et à se familiariser aux règles de développement aux travers de petites tâches (de la conception UML à la création d'écrans et de processus en passant par la génération de code).
La plus grosse partie de mon travail s'est faîte sur un module spécifique à un client, le module Lola.
Ce client souhaitait utiliser OfbizNégia derrière un portail web tout en faisant des requêtes vers des moteurs de recherche (VOL, HOTEL, SEJOUR, etc ...). Ce besoin particulier, nécessitant des développements en grande partie spécifiques, un cvs à été mis en place.
Le shéma ci-dessous va nous aider à mieux comprendre le fonctionnement et le rôle de chaque acteur :



Le portail se charge de toute la partie graphique et uniquement de cette partie. OfbizNégia reçoit les requêtes du portail, les traite et lui renvoit sa réponse sous forme d'un flux xml. OfbizNéogia possède un service de recherche de haut niveau en java qui permet d'interroger les différents moteurs de recherche nécessaires. Les requêtes aux moteurs se font par l'intermédaire de Tibco. Il s'agit d'une entreprise qui gère et fournit les services pour interroger les moteurs. Cependant lors du dévellopement l'accès à Tibco n'étant pas encore disponible, il a été nécessaire de développer une application simulant son rôle. TibcoSim a été développée par Peter Goron dans ce but.
La communication entre Tibco/TibcoSim et OfbizNégia est réalisée par l'intermédaire d'un Middleware, JMS. L'interface de programmation Java Message Service (JMS) permet d'envoyer et de recevoir des messages de manière asynchrone entre applications ou composants Java. JMS permet l'échange de messages entre deux systèmes ou plus. Ce service supporte le modèle publish and subscribe et le modèle point à point. Dans le modèle publish and subscribe, des entités s'inscrivent pour recevoir des messages sur un certain sujet. Et celui qui publie les messages et ceux qui les reçoivent ne se connaissent pas. Pour le modèle point à point, le producteur publie les messages dans une file et le client lit les messages de la file. Dans ce cas le producteur connaît la destination des messages et poste les messages directement dans la file du client. Pour simuler le rôle de Tibco, une implémentation libre de JMS, OpenJms a été mis en place pour permettre la communication avec OfbizNéogia.

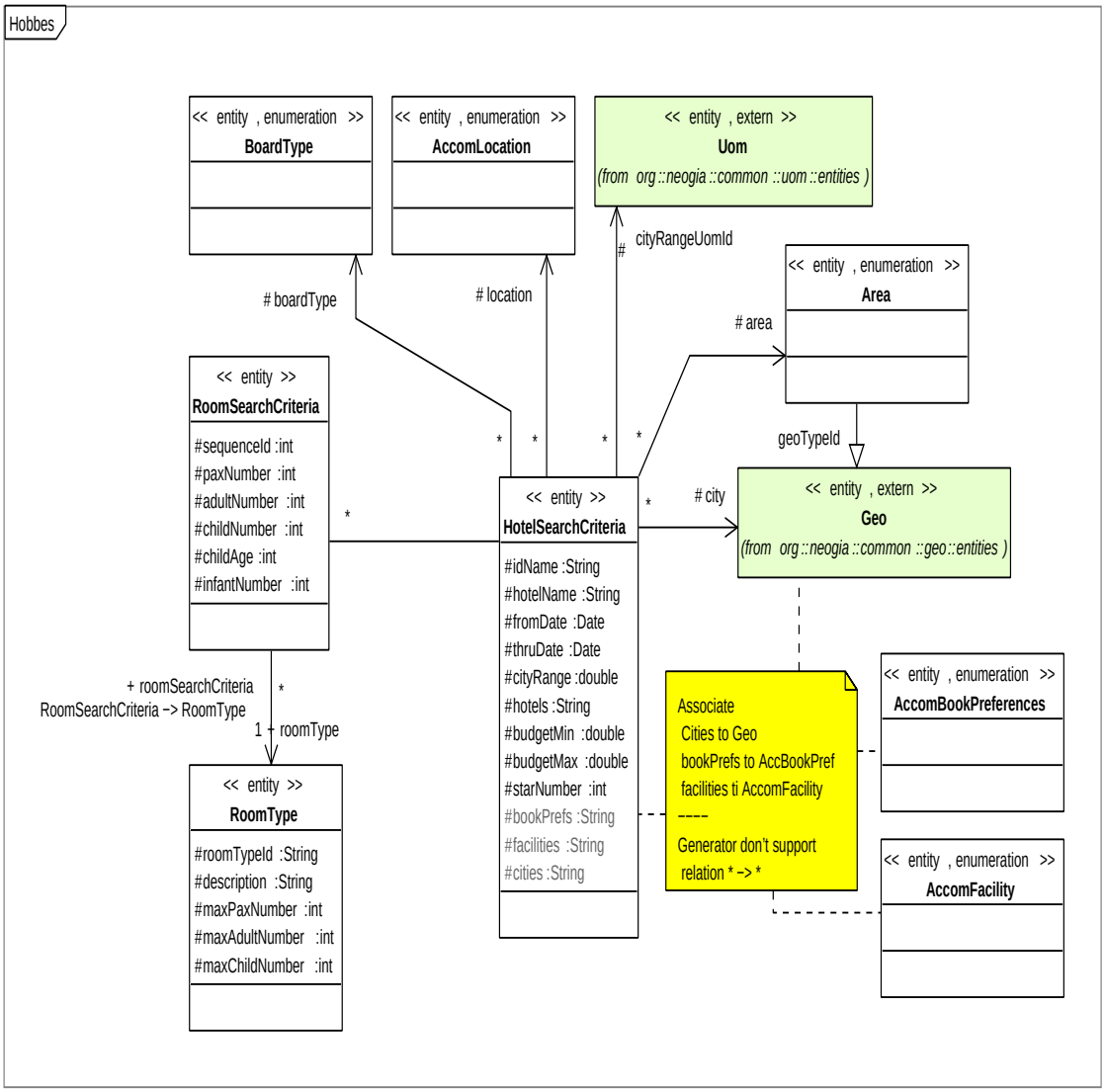
3.3.2 Modélisation UML

A partir de la documentation fonctionnel fournie par le client, une première modélisation a été effectuée. Il s'agit a cet étape de réaliser le mapping entre les entités de Néogia et celle nécessaires aux besoins du client. Certaines entités comme les acteurs (module Party) sont facilement intégrables à Néogia car déjà présentes. Par contre il y a bien entendu des besoins spécifiques qui impliquent la création de nouvelles entités.
Les diagrammes de classe suivants ont été conçus dans ce but en respectant les règles de modélisation impératives à une génération de code propre :

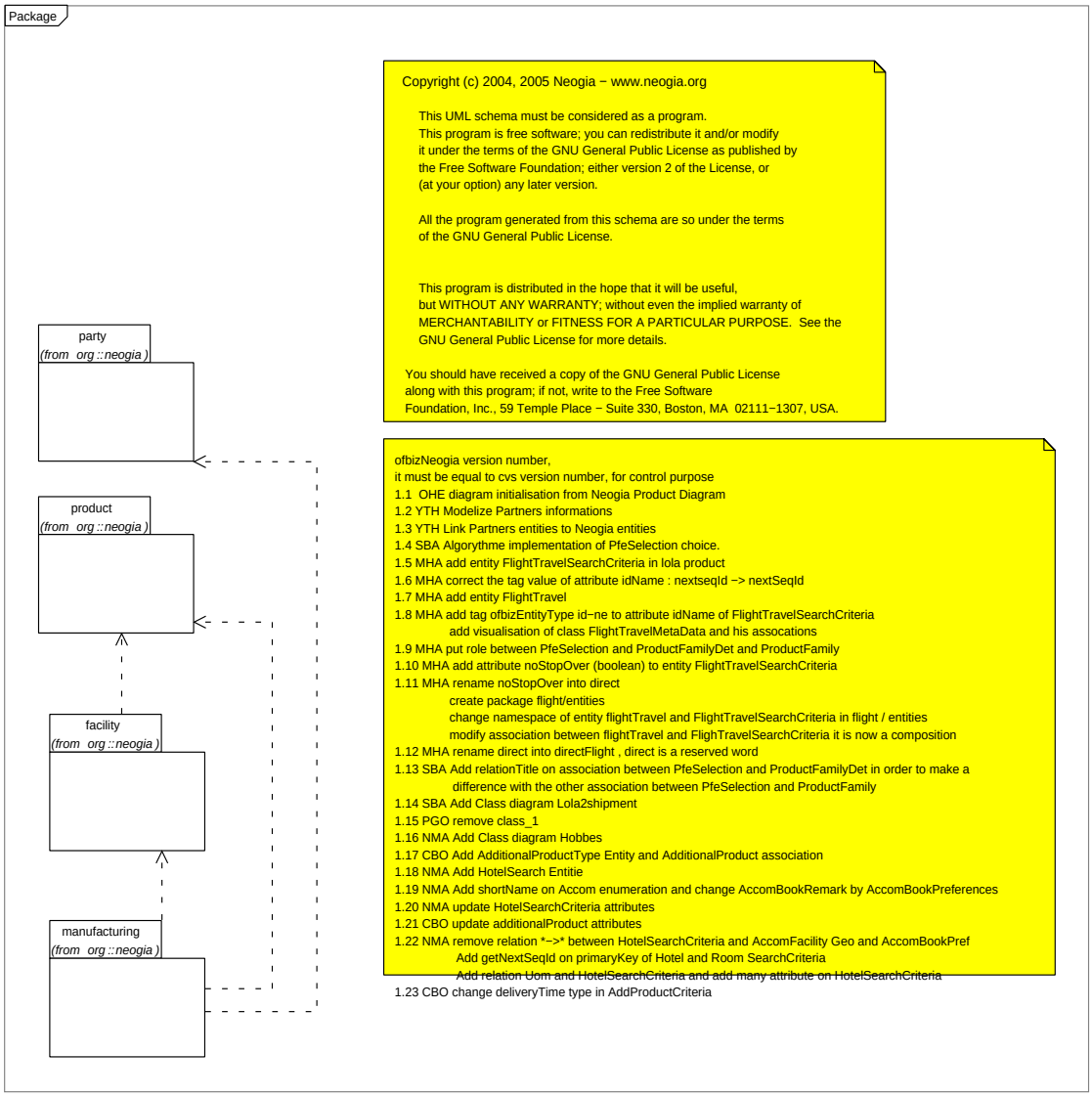


Ce diagramme permet de "mapper" la réponse de Tibco à une requête au moteur VOL (Partners). Certaines entités comme l'entité "Party" ont en plus du stéréotype "entity", le stéréotype "extern" qui signifie que ce sont des entités déjà définies dans une autre modélisation. Dans le cas de "Party", l'entité est déjà définie dans le module du même nom. Le générateur se charge lui même de faire le lien entre les différentes modélisations. L'entité "FlightTravelSearchCriteria" par enxemple, quant à elle est spécifique au module Lola. Plus précisement, ce diagramme permet d'interpréter la BasicAPI vue dans le shéma de description précédent. Ce document xml décrit les entités crée dans le diagramme de classe partners.

De même que pour le moteur VOL (Partners) un diagramme de classe réalise le mapping entre la réponse à une réquête sur ce moteur et ofbizNeogia :



Un diagramme commun à tous les modèles UML des modules, est le diagramme "Package" qui décrit les liens entre les packages mais surtout qui permet de garder une trace de toutes les modifications appliquées au modèle du module avec les numéros de version.



Une fois les diagrammes créés, on passe dans le répertoire du module, puis on lance la commande ant. L'exécution lit le fichier UML puis lance la génération des fichiers. Une fois l'exécution de cette dernière finie, on obtient une première liste de fichiers pour le module.
Les fichiers générés ne sont pas tous utilisables. Il y a, en gros, trois types de fichier :

- les fichiers utilisables directement après la génération : tous les fichiers ftl, de form, bsh, sources, de description des entités .
- les fichiers à valider soi même : les fichiers de définition de page et les fichiers sources contenant des objets hérités
- les fichiers qui nécessitent un transfert du contenu manuellement : service.xml et controller.xml

Afin de distinguer les fichiers générés des fichiers développés, dans chaque répertoire du module pouvant avoir ces deux types de fichiers se trouvent deux répertoires : "developed" et "generated" qui servent à organiser le développement.

3.3.3 Création de l'arborescence du module

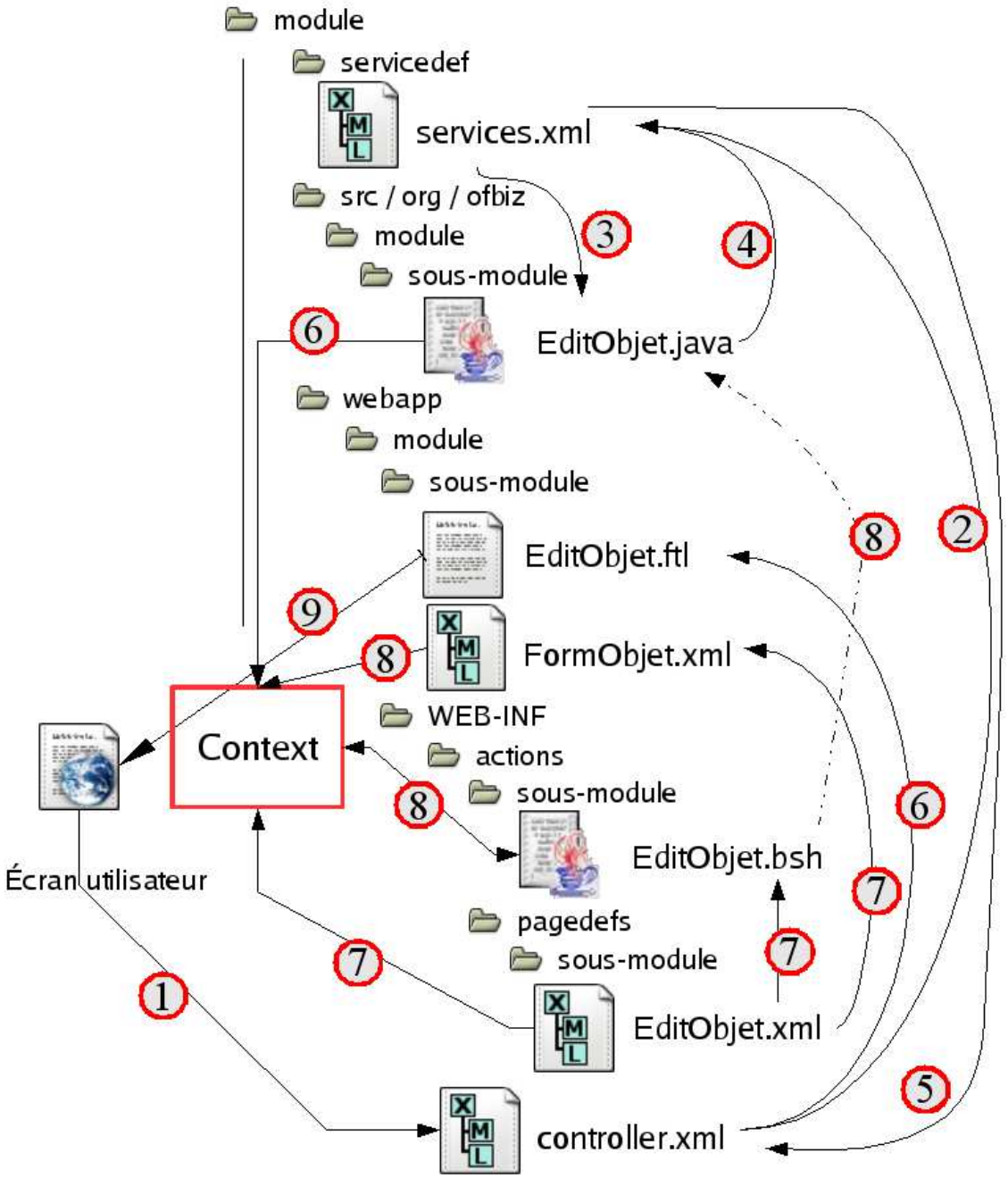
La première tâche, une fois les fichiers générés, consiste à savoir de quels fichiers on a besoin. Les fichiers contenus dans les répertoires "generated" et les répertoires crées sont automatiquement copiés par le générateur dans le module correspondant. Les fichiers des répertoires "developed" et certains autres fichiers comme ceux définissant les entités (dans le répertoire "modeldef") ne sont copiés qu'une fois, à la première génération. Pour les générations suivantes il faudra copier soit même les fichiers dont ont a besoin.
L'arborescence de Lola se présente comme ceci :

```
lola
|
+- build
|
+- config
|
+- data
|
+- doc
|
+- entitydef
|
+- lib
|
+- modeldef
|
+- script
|
+- servicedef
|
+- src
|
+- webapp
|
+- webcommon
|
+- widget
|
+- xsd
```

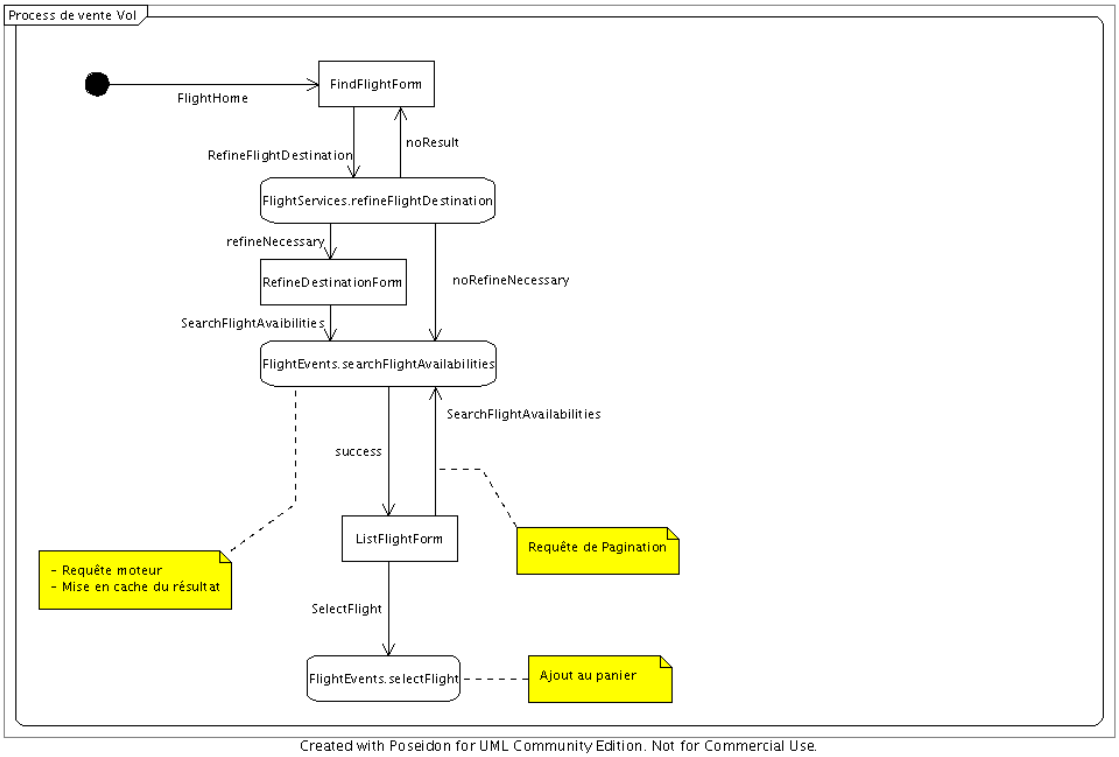
Une fois l'arborescence faite, vient le développement spécifique.

3.3.4 Dévellopement des écrans

Pour chaque écran du portail, un flux xml correspondant doit être retourné. Le portail fait donc une requête html à OfbizNéogia. Pour illustrer ce développement nous allons suivre un exemple que j'ai développé.
Il s'agit de suite d'écrans de recherche de vols.
Voyons tout d'abord le cheminement d'une requête au serveur OfbizNéogia :



L'enchainement des requêtes et des écrans (flux xml) est fait comme ceci :



3.3.5 Le fichier "controller.xml"

La première tâche est de remplir le fichier "controller.xml". A la réception de la requête, le serveur lit ce fichier du module concerné pour savoir ce qui doit être exécuté. Il faut donc définir les requêtes délivrant les flux xml correspondant à la page de recherche des vols.

```
- <request-map uri="FlightHome">
    <security https="false" auth="false"/>
    <response name="success" type="view" value="FlightHome"/>
  </request-map>
- <request-map uri="FlightHomeHtml">
  - <!--
      uri to use to test with admin-html : http://127.0.0.1:8080/lola/control/FlightHomeHtml?USERNAME=admin-html&PASSWORD=ofbiz
    -->
    <security https="false" auth="false"/>
    <event type="java" path="org.ofbiz.securityext.login.LoginEvents" invoke="login"/>
    <response name="success" type="view" value="FlightHome"/>
  </request-map>
- <request-map uri="RefineFlightDestination">
    <security https="false" auth="false"/>
    <event type="service" invoke="refineFlightDestination"/>
    <response name="noResult" type="view" value="FlightHome"/>
    <response name="refineNecessary" type="view" value="RefineFlightDestination"/>
    <response name="noRefineNecessary" type="view" value="ListFlight"/>
  </request-map>
- <request-map uri="SearchFlightAvailabilities">
    <security https="false" auth="false"/>
    <event type="java" path="org.ofbiz.lola.flight.developed.FlightEvents" invoke="getFlightTravel"/>
    <response name="success" type="view" value="ListFlight"/>
  </request-map>
```

La "RequestMap" "FlightHome" définit la page d'accueil de la recherche des vols, "FlightHomeHtml" sert quant à elle à tester la réponse du serveur en html afin de vérifier et de valider le développement. La réponse à effectuer y est décrite :
<response name="success" type="view" value="FlightHome"/>
Dans ce même fichier, sont définies les "view-map" correspondantes aux réponses :

```
- <!--
    ============================================================
  -->
- <!--
                  Begin of Flight-related view-map
  -->
- <!--
    ============================================================
  -->
  <view-map name="FlightHome" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#FindFlight"/>
  <view-map name="ListFlight" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#ListFlight"/>
  <view-map name="ShowFlightSelection" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#ShowFlightSelection"/>
  <view-map name="ShowFlightSelected" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#ShowFlightSelected"/>
  <view-map name="StoreFlightTravel" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#StoreFlightTravel"/>
  <view-map name="ListFlightProduct" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#ListFlightProduct"/>
  <view-map name="RefineFlightDeparture" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#RefineFlightDeparture" content-type="text/xml"/>
  <view-map name="RefineFlightDestination" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#RefineFlightDestination" content-type="text/xml"/>
```

On y indique le flux xml à générer via le fichier xml indiqué :

```
<view-map name="FlightHome" type="xml" page="component://lola/widget/lola/flight/developed/FlightScreens.xml#FindFlight"/>
```

L'option "type" indique qu'il s'agit d'un flux xml et "page" donne le chemin vers sa définition. Les flux xml sont générés via un handler.

Tous les "handler" nécessaires sont définis en tête du fichier "controller.xml" :

```
<site-conf xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/site-conf.xsd">
  <description>LOLA: eCommerce Controller Configuration File</description>
  <owner>Neogia.org</owner>
  <errorpage>/error/error.jsp</errorpage>
  <handler name="java" type="request" class="org.ofbiz.webapp.event.JavaEventHandler"/>
  <handler name="bsf" type="request" class="org.ofbiz.webapp.event.BsfEventHandler"/>
  <handler name="soap" type="request" class="org.ofbiz.webapp.event.SOAPEventHandler"/>
  <handler name="service" type="request" class="org.ofbiz.webapp.event.ServiceEventHandler"/>
  <handler name="service-multi" type="request" class="org.ofbiz.webapp.event.ServiceMultiEventHandler"/>
  <handler name="simple" type="request" class="org.ofbiz.webapp.event.SimpleEventHandler"/>
  <handler name="ftl" type="view" class="org.ofbiz.webapp.ftl.FreeMarkerViewHandler"/>
  <handler name="jsp" type="view" class="org.ofbiz.webapp.view.JspViewHandler"/>
  <handler name="http" type="view" class="org.ofbiz.webapp.view.HttpViewHandler"/>
  <handler name="screen" type="view" class="org.ofbiz.widget.screen.ScreenWidgetViewHandler"/>
  <handler name="xml" type="view" class="org.ofbiz.widget.GenericScreenViewHandler"/>
  <handler name="simplecontent" type="view" class="org.ofbiz.content.view.SimpleContentViewHandler"/>
  <handler name="xmlhttp" type="view" class="org.ofbiz.webapp.view.XmlHttpViewHandler"/>
```

3.3.6 Le fichier "FlightScreens.xml"

Voyons maintenant le fichier "FlightScreens.xml" et plus précisément la défintion de l'écran "FlightHome" :

```
<screen name="FindFlight">
 - <section>
   - <actions>
       <set field="titleProperty" value="PageTitleFindFlight"/>
     </actions>
   - <widgets>
     - <decorator-screen name="main-decorator-neogia" location="component://lola/widget/lola/CommonScreens.xml">
       - <decorator-section name="body">
           <include-form name="FindFlightForm" location="component://lola/widget/lola/flight/developed/FlightForms.xml"/>
         </decorator-section>
       </decorator-screen>
     </widgets>
  </section>
</screen>
```

Ce fichier contient la définition de tous les écrans de recherche des vols. Chaque "screen" est décomposé en "section" qui elles même sont composées d'une partie "actions" et d'une partie "widgets". La première permet de réaliser des actions sur l'écran comme par exemple l'appel d'un service java ou d'un fichier beanshell (java interprété) afin d'affecter des valeurs aux différents champs de l'écran. La partie "widgets" contient des balises "decorator-screen" et "decorator-section" qui définissent la décoration générale de l'écran. Dans le cas du module Lola, les décorateurs ont été alégés pour ne pas surcharger les flux xml de réponse.
Enfin chaque "screen" inclut une ou plusieurs "form". C'est le rôle de la balise "include-form" :

```
<include-form name="FindFlightForm" location="component://lola/widget/lola/flight/developed/FlightForms.xml"/>
```

3.3.7 Le fichier "FlightForms.xml"

De même que pour le fichier "FlightScreens.xml", le fichier "FlightForms" contient toutes les "form" nécessaire à la recherche des vols. Les "form" définissent les champs des écrans et dans notre cas des flux xml.
Voici une partie de la "form" qui décrit la page d'accueil de la recherche des vols :

```
<forms xsi:noNamespaceSchemaLocation="../../../../../framework/widget/dtd/widget-form.xsd">
- <!--
        * Formulaire de recherche spécialisé pour les vols
        *
        * status : delivered
  -->
- <form name="FindFlightForm" type="single" target="RefineFlightDestination">
  - <field name="productCategoryId">
    - <radio no-current-selected-key="LOLA_FLIGHT_FAMILY">
        <option key="LOLA_FLIGHT_FAMILY" description="Un vol"/>
        <option key="LOLA_HOTEL_FAMILY" description="Un hôtel"/>
        <option key="LOLA_TO_FAMILY" description="Un séjour"/>
        <option key="LOLA_RENT_FAMILY" description="Une voiture"/>
      </radio>
    </field>
  - <field name="adult" title="Adultes" position="1" tooltip="Choisissez le nombre de voyageurs adultes">
    - <drop-down no-current-selected-key="2" current="selected" allow-empty="false">
      - <entity-options entity-name="ProductFeature" key-field-name="value" description="${description}">
          <entity-constraint name="productFeatureCategoryId" operator="equals" value="LOLA_FLIGHT_ADU_PASS"/>
          <entity-order-by field-name="defaultSequenceNum"/>
        </entity-options>
      </drop-down>
    </field>
  - <field name="child" title="Enfants" position="2" tooltip="Choisissez le nombre de voyageurs entre 2 et 11 ans">
    - <drop-down no-current-selected-key="0" current="selected" allow-empty="false">
      - <entity-options entity-name="ProductFeature" key-field-name="value" description="${description}">
          <entity-constraint name="productFeatureCategoryId" operator="equals" value="LOLA_FLIGHT_CHD_PASS"/>
          <entity-order-by field-name="defaultSequenceNum"/>
        </entity-options>
      </drop-down>
    </field>
  - <field name="infant" title="Bébés" position="3" tooltip="Choisissez le nombre de voyageurs entre 0 et 2 ans">
    - <drop-down no-current-selected-key="0" current="selected" allow-empty="false">
      - <entity-options entity-name="ProductFeature" key-field-name="value" description="${description}">
          <entity-constraint name="productFeatureCategoryId" operator="equals" value="LOLA_FLIGHT_INF_PASS"/>
          <entity-order-by field-name="defaultSequenceNum"/>
        </entity-options>
      </drop-down>
    </field>
  - <field name="departure" title="Départ" position="1" tooltip="Choisissez votre ville de départ préférée">
```

Dans le module lola il est possible d'utiliser les "form" de type "single" et "list" car le handler xml sait les traiter. Chaque "field" correspond à un champ de l'écran et il est possible d'y afficher toutes sortes de données, d'une simple valeur texte à une liste de valeurs contenues dans des entités de la base de données du serveur OfbizNéogia. Il est aussi possible de récupérer des variables stockées dans le contexte. Maintenant que nous avons vu un exemple simple, voyons un autre exemple des tâches que j'ai eu à réaliser qui nécessite l'utilisation des services et évènements java.

3.3.8 Dévellopement des processus

L'exemple précédent d'une "request-map" n'utilisait pas ni d'évènement ni de services java. Les exemples suivants illustrent leur utilisation :

```
<request-map uri="RefineFlightDestination">
    <security https="false" auth="false"/>
    <event type="service" invoke="refineFlightDestination"/>
    <response name="noResult" type="view" value="FlightHome"/>
    <response name="refineNecessary" type="view" value="RefineFlightDestination"/>
    <response name="noRefineNecessary" type="view" value="ListFlight"/>
</request-map>
<request-map uri="SearchFlightAvailabilities">
    <security https="false" auth="false"/>
    <event type="java" path="org.ofbiz.lola.flight.developed.FlightEvents" invoke="getFlightTravel"/>
    <response name="success" type="view" value="ListFlight"/>
</request-map>
```

Les services

Pour la "resquest-map" "RefineFlightDestination" dont le but est d'affiner la destination souhaitée pour le vol recherché, il peut y avoir trois réponses différentes suite à l'exécution du service java *<event type="service" invoke="refineFlightDestination"/>* Dans le cas où la destination n'a pas de correspondance, le service retourne le message "noResult" et le flux xml correspondant à l'écran d'accueil "FlightHome" est renvoyé. Si la destination a une seule correspondance, le message est "noRefineNecessary" et le flux "ListFlight" qui liste les résultats de la recherche est renvoyé. Enfin si il existe plusieurs choix possibles pour la destination, le message "refineNecessary" est retourné par le service et un flux correspondant à un écran de choix "RefineFlightDestination" est envoyé au portail.
Avant de développer à proprement parler le service, il faut déjà le définir dans le fichier "/service-def/developed/services.xml" :

```
<service name="getProdCatalog" engine="java" location="org.ofbiz.lola.webSite.developed.WebSiteServices" invoke="getProdCatalog" auth="false">
    <description>Get ProdCatalog details list</description>
    <attribute name="productStoreId" mode="IN" type="String" optional="false"/>
    <attribute name="prodCatalogId" mode="IN" type="String" optional="true"/>
    <attribute name="prodCatalogDetailsList" type="java.util.List" mode="OUT"/>
</service>
<service name="listShipmentMeth" engine="java" location="org.ofbiz.shipment.shipment.developed.ShipmentServices" invoke="getListProductStoreShipmentMeth" auth="false">
    <description>List shipment method</description>
    <attribute name="inputFields" type="java.util.Map" mode="IN" optional="true"/>
    <attribute name="listIt" type="java.util.List" mode="OUT" optional="true"/>
</service>
<service name="refineFlightDestination" engine="java" default-entity-name="FlightTravelSearchCriteria" location="org.ofbiz.lola.flight.developed.FlightServices" invoke="refineFlightDestination"
auth="false">
    <description>Refine flight parameters</description>
    <auto-attributes include="all" mode="INOUT" optional="true"/>
    <attribute name="flightListDestinationResult" type="java.util.List" mode="OUT" optional="true"/>
</service>
<service name="CreateUrgencyContact" engine="java" location="org.ofbiz.lola.myaccount.developed.MyAccountServices" invoke="createUrgencyContact" auth="true">
    <description>Create Urgency Contact</description>
    <attribute name="titleProfile" mode="IN" type="String" optional="false"/>
    <attribute name="firstName" mode="IN" type="String" optional="false"/>
    <attribute name="lastName" mode="IN" type="String" optional="false"/>
    <attribute name="prefixTel" mode="IN" type="String" optional="true"/>
    <attribute name="prefixMobile" mode="IN" type="String" optional="true"/>
    <attribute name="numberTel" mode="IN" type="String" optional="true"/>
    <attribute name="numberMobile" mode="IN" type="String" optional="true"/>
```

Parmis d'autres définitions de services, le service "refineFlightDestination" est défini avec des paramêtres en entrée et en sortie. L'option "default-entity-name" permet de lier les paramêtres d'entrée avec les attributs d'une entité. Ainsi le champ "auto-attributes" signifie que tous les attributs de l'entité "FlightTravelSearchCriteria" font partie des paramêtres d'entrée et de sortie "mode="INOUT"" et qu'il ne sont pas obligatoire "optional="true"". Une liste de résultats est envoyé en paramêtre de sortie et par défaut un message associé à la réponse peut être renvoyé.

Algorithme d'affichage de la destination

```
Entrée = destination :String
Sortie = message :String + listDestination :List

List listDestination <- listeVide
Si(destination == null)Alors
| retourner "noResult"
FinSi
Si(isAnAirport(destination))Alors
| listDestination <- destination
Sinon
| Si(isACity(destination))Alors
| | listDestination <- ListeAirportCity(destination) ;
| Sinon
| | Si(isACountry(destination))Alors
| | | List listCity <- ListeCityCountry(destination) ;
| | | TantQue( !Fin(listCity))Alors
| | | | listDestination <- ListeAirportCity(listCity.extract()) ;
| | | FinTantQue
| | FinSi
| FinSi
FinSi
Entier taille <- listDestination.size()
Si(taille == 0)Alors
| retourner "noResult"
FinSi
Si(taille == 1)Alors
| retourner "noRefineNecessary"
FinSi
Si(taille > 1)Alors
| Si(taille > 10)Alors
| | listDestination <- TrierRéduireTaille(listdestination)
| FinSi
| retourner ("refineNecessary" + listDestination)
FinSi
```

Voici un extrait du service java "refineFlightDestination" qui valide la destination du vol, propose une liste de choix possibles ou renvoit à la page d'accueil si aucune correpondances avec des destinations connues dans la base n'ont été trouvées.

```
...
public class FlightServices {
    private static final String module = FlightServices.class.getName();

    public static Map refineFlightDestination(DispatchContext ctx, Map context) {
        //4 cas possibles destination=un aÃ©roport, destination=une ville,
  //destination=pays/rÃ©gion, destination ne corespond Ã  aucun geo
        GenericDelegator delegator = ctx.getDelegator();
        Locale locale = (Locale) context.get("locale");
        Map result = new FastMap();
        if(context.get("destination") == null){
            result.put(ModelService.RESPONSE_MESSAGE, "noResult");
            result.put(ModelService.ERROR_MESSAGE, "no corresponding destination");
            return result;
        }
        String destination = ((String)context.get("destination")).toUpperCase(locale);
        List flightListDestinationResult = new FastList();

        //1er cas, destination est un aÃ©roport
        Map tmp = isAnAirport(destination, delegator);
        String response = (String)tmp.get("response");
        if(response.equals("true")){
            Geo airport  = (Geo)tmp.get("airport");
            FlightGeo flightGeo = new FlightGeo(airport.getGeoId(),airport.getGeoName()
,airport.getGeoCode(),(String)airport.get("geoTypeId"),0);
            flightListDestinationResult.add(flightGeo);
        }else
        {
            //2eme cas, destination est une ville
    ...

        if(flightListDestinationResult.isEmpty()){
            result.put(ModelService.RESPONSE_MESSAGE, "noResult");
            result.put(ModelService.ERROR_MESSAGE, "no corresponding destination");
        }
        if(flightListDestinationSize == 1){
            result.put("destination",(String)((FlightGeo)flightListDestinationResult.
listIterator().next()).getGeoId());
            result.put(ModelService.RESPONSE_MESSAGE, "noRefineNecessary");
            result.put(ModelService.ERROR_MESSAGE, "one corresponding destination");
        }
        if(flightListDestinationSize > 1){
            //trie de la liste pour sortir les dix meilleurs rÃ©sultats
            if(flightListDestinationSize > 10){
               flightListDestinationResult  = sortListResult(flightListDestinationResult);
            }
            result.put(ModelService.RESPONSE_MESSAGE, "refineNecessary");
            result.put(ModelService.ERROR_MESSAGE, "severals corresponding destinations");
            result.put("flightListDestinationResult",flightListDestinationResult);
        }
        return result;
    }
```

Les évènements

Contrairement aux services java qui sont enregistrés sur le serveur, les évènements sont surtout utilisés pour appeler des services comme dans l'exemple suivant, la ligne en gras est un appel au service "getFlightAvailabilities" avec en entrée une "Map" contenant les paramêtres.

```
public class FlightEvents {
    private static final String module = FlightEvents.class.getName();

    public static final String FLIGH_SEARCH_KEY = "flighLastSearchKey";
    public static final String FLIGH_AVAILABILITIES_KEY = "flighLastSearchResult";

    public static String getFlightTravel(HttpServletRequest request, HttpServletResponse response) {
        LocalDispatcher dispatcher = (LocalDispatcher) request.getAttribute("dispatcher");
        GenericValue userLogin = (GenericValue) request.getAttribute("userLogin");
        Locale locale = UtilHttp.getLocale(request);
        HttpSession session = request.getSession();

...

        Map result = null;
        try {
            Map serviceContext = UtilMisc.toMap("flightTravelSearchCriteria", flightTravelSearchCrit
            result = dispatcher.runSync("getFlightAvailabilities", serviceContext);
        } catch (GenericServiceException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return "error";
        }

        if (!(ServiceUtil.isError(result) || ServiceUtil.isFailure(result))) {
            List flightTravelList = (List) result.get("flightTravelList");
            session.setAttribute(FlightEvents.FLIGH_SEARCH_KEY, searchKey);
            session.setAttribute(FlightEvents.FLIGH_AVAILABILITIES_KEY, flightTravelList);
        }

        // set the messages in the request; this will be picked up by messages.ftl and displayed
        request.setAttribute("_ERROR_MESSAGE_LIST_", result.get(ModelService.ERROR_MESSAGE_LIST));
        request.setAttribute("_ERROR_MESSAGE_MAP_", result.get(ModelService.ERROR_MESSAGE_MAP));
        request.setAttribute("_ERROR_MESSAGE_", result.get(ModelService.ERROR_MESSAGE));

        request.setAttribute("_EVENT_MESSAGE_LIST_", result.get(ModelService.SUCCESS_MESSAGE_LIST));
        request.setAttribute("_EVENT_MESSAGE_", result.get(ModelService.SUCCESS_MESSAGE));

        return (String) result.get(ModelService.RESPONSE_MESSAGE);
    }
...
```

Les fichiers "beanshell"

Dans la partie "action" de la définition des "screen", du code "beanshell" peut être exécuté. Il s'agit de code java interprété.
Durant l'exécution des fichiers bsh, diverses informations sont inscrites dans le contexte. Ces dernières peuvent provenir des résultats d'une recherche dans la base de données, d'un appel de fonction ou encore de l'utilisation d'algorithme.
L'exemple qui suit montre le "screen" qui propose une liste de destinations possibles et qui fait appel à un script ".bsh"

```
<screen name="RefineFlightDestination">
- <section>
  - <actions>
      <set field="viewSize" from-field="parameters.VIEW_SIZE" default-value="20" type="Integer"/>
      <set field="viewIndex" from-field="parameters.VIEW_INDEX" type="Integer"/>
      <script location="component://lola/script/widgetActions/flight/RefineFlightDestination.bsh"/>
    </actions>
  - <widgets>
    - <decorator-screen name="main-decorator-neogia" location="component://lola/widget/lola/CommonScreens.xml">
      - <decorator-section name="body">
          <include-form name="refineDestinationForm" location="component://lola/widget/lola/flight/developed/FlightForms.xml"/>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

Le script ".bsh" correspondant :

```
import java.util.Map;
import java.util.List;
import java.util.ListIterator;

import javolution.util.FastMap;
import javolution.util.FastList;
import org.ofbiz.entity.GenericValue;
import org.ofbiz.base.util.UtilMisc;
import org.ofbiz.lola.flight.ComparatorCityAirport;
import org.ofbiz.lola.flight.FlightGeo;

List flighListDestinationResultBefore = (List)parameters.get("flightListDestinationResult");
List flighListDestinationResultAfter = new FastList();
int count = 0;

if(flighListDestinationResultBefore != null){
    ListIterator listIt = flighListDestinationResultBefore.listIterator();
    while(listIt.hasNext()){
        count = count+1;
        Map maMap = new FastMap();
        FlightGeo flightGeo = (FlightGeo)listIt.next();
        maMap.put("geoId",flightGeo.getGeoId());
        if(flightGeo.getGeoTypeId().equals("LOLA_CITY_CODE"))
            maMap.put("geoName",flightGeo.getGeoName()+" tous les aeroports ");
        else
            maMap.put("geoName",flightGeo.getGeoName());
        maMap.put("geoCode","("+flightGeo.getGeoCode()+")");
        flighListDestinationResultAfter.add(maMap);
    }
}
context.put("flightListDestinationResult",flighListDestinationResultAfter);
```

3.3.9 Exemple de flux xml

Le résultat de la recherche des vols se présente sous la forme d'un flux xml alégé.

```
<?xml version="1.0" encoding="UTF-8"?>
<screen:screen xmlns:form="http://www.neogia.org/xsd/xml-widget-form" xmlns:screen="http://www.neogi
xsd/xml-widget-screen" name="ShowFlightSelected">
  <form:single-form name="showFlightSelected" title="OPTIONS ET MODE DE LIVRAISON" target="/lola/con
StoreFlightTravel;jsessionid=330D446424E8553B5D7D01861CF3BAD5.jvm1">
      <form:field-group>
      <form:field name="flight">
        <form:display>
          <form:value>Vol PARIS - LONDON</form:value>
        </form:display>
       </form:field>
      <form:field name="nbPassengers">
        <form:display>
          <form:value>2 adulte(s)  A</form:value>
        </form:display>
       </form:field>
      </form:field-group>
      <form:field-group>
      <form:field name="dateHourDepartureGo">
        <form:display>
          <form:value>22 janv. 2007 &amp;agrave; 12:10:00</form:value>
        </form:display>
       </form:field>
      <form:field name="dateHourDepartureBack">
        <form:display>
          <form:value>22 janv. 2007 &amp;agrave; 12:20:00</form:value>
        </form:display>
       </form:field>
      <form:field name="company">
        <form:display>
          <form:value>easyJet</form:value>
        </form:display>
       </form:field>
      <form:field name="basePrice">
        <form:display>
          <form:value>149.6748&amp;euro; / adulte </form:value>
        </form:display>
       </form:field>
      </form:field-group>
      <form:field-group>
      <form:field name="passengerPrice">
        <form:display>
          <form:value>299.3496 &amp;euro; TTC</form:value>
        </form:display>
       </form:field>
      <form:field name="serviceCost">
        <form:display>
          <form:value>+35.00  &amp;euro; TTC</form:value>
        </form:display>
       </form:field>
      </form:field-group>
      <form:field name="info1">
        <form:display>
          <form:value/>
        </form:display>
```

```
      </form:field>
     <form:field name="info2">
       <form:display>
         <form:value/>
       </form:display>
      </form:field>
</form:single-form>
...
```

Chapitre 4

Bilan

4.1 Bilan professionnel

Le premier point positif de ce stage a été pour moi de travailler sur des logiciels libres. J'ai pu avoir un bon aperçu des méthodes de travail en entreprise. Le travail sur un ERP m'a permis de d'aborder et d'approfondir un grande nombre de thèmes reliés au monde de l'entreprise, tant dans le domaine technique que fonctionnel.
Outre ce côté là, les développements dans un environnement J2EE m'ont permis d'acquerir une certaine expérience dans le domaine des architectures logicielles. j'ai également pu me familiariser avec un large panel de technologies de haut niveau comme la génération de code et de prendre conscience de l'importance du travail en équipe.

4.2 Bilan personnel

Sur le plan personnel j'ai pu approcher les développements d'applications libres, leur fonctionnement et surtout la communauté qui entoure ces projets.
Le fait de travailler dans une société de service spécialisée dans le logiciel libre a été pour moi un moyen de d'aborder le monde du libre et d'en conforter mon intérêt et mes connaissances. J'ai aussi pu profiter de la convivialité de l'équipe Néréide pour m'intéresser à leur expérience et leur parcours respectif.

Conclusion

OfbizNéogia est un ERP très complex et qui nécessite une compréhension globale afin de développer correctement les tâches. En effet il arrive fréquement qu'il faille adopter une vision générique ce qui entraine généralement de retoucher des modules autres que celui sur lequel on travaille. Pour celà le début du stage n'a pas été toujours évident aux vues du nombre de nouveaux outils et de technologies qu'il a été nécessaire d'apprendre à utiliser et à comprendre. Cependant les tâches qui m'ont été assignées ont été accomplies dans l'enssemble.

Ce stage m'a permis de me familiariser avec les rêgles techniques exigentes dans des développements complexes comme ceux qui sont effectués sur OfbizNéogia et qui nécessitent de bonnes connaissances avant de pouvoir effectuer un travail éfficace.
Enfin, mis à part l'apport technique, ce stage m'a ouvert à la dynamique du logiciel libre.

Résumé : Durant ce stage j'ai réalisé des développements sur l'ERP libre OfbizNeogia. J'ai appris à utiliser et j'ai utilisé des technologies diverses et complexes telles que l'environnement J2EE, le java interprété (beanshell), freemarker, ou encore la génération de code.

Mots clé : Ofbiz, Neogia, ERP, Génération de code, java, J2EE, beanshell, xml, freemarker.

Abstract : During this training course I carried out developments on the free ERP OfbizNeogia. I learned how to use and I used various and complex technologies such as interpreted environment J2EE, the java (beanshell), freemarker, or the code generation.

Keywords : Ofbiz, Neogia, PGI, Code generation, java, J2EE, beanshell, xml, freemarker.

Encadrants :

Olivier HEINTZ

Etudiants :

Simon BAUDRY

2^{eme} Année - 2005-2006