
# Découverte du framwork Apache OFBiz. Dévéloppement d'une API HTTP, basé sur le style architectural REST et inégration dans un contexte de projet client.

*Auteur :*
Artemiy Rozovyk

*Tuteur de stage :*
Mathieu Lirzin
*L'enseignant référent :*
Florent Foucaud

1er juin 2019

# Table des matières

# Remerciements

Merci tout le monde !

# Chapitre 1

# Introduction

Le présent document expose le travail effectué lors du stage de fin de licence au sein de la société Néréide. Ce stage se décompose en deux parties : Premièrement, il a pour but de se familiariser avec la suite d'applications libres pour l'entreprise Apache OFBiz et son utilisation dans le contexte de la société d'accueil. Finalement, il consiste à intégrer un système permettant la définition des API HTTP du style REST, ainsi que la modification d'une API existante afin de fournir une preuve de concept consistante.
Le travail a été effectué en étroite collaboration avec le principal concerné : la communauté Apache, ce qui a contribué à une meilleure cohérence entre le travail réalisé et les besoins des utilisateurs.
Dans un premier temps nous allons présenter l'entreprise d'accueil ainsi que faire une description de l'outil principal utilisé. Dans un deuxième temps nous exposerons la démarche qui a permis une compréhension suffisante du Framework OFBiz nécessaire à la partie finale du stage, REST, qui sera décrite dans un troisième temps.

# Chapitre 2

# Contexte du stage

Intro

## 2.1 Entreprise

### 2.1.1 Presentation générale

Néréide est une société de services en logiciels libres crée en 2004 qui se spécialise dans l'intégration du progiciel de gestion intégré Apache OFBiz. Il s'agit d'une société coopérative et participative (SCOP) qui se situe à Tours.

### 2.1.2 Activité

Transparence libre entreprise

Integration(utilisation des briques en tant que telles) et Dévéloppement spécifique (adaptation d'OFBIZ pour les besoins ) plugins propres à la logique métier. Pas de forfait(contrat doit etre rempli, l'entreprise s'engage à ce qu'il soit livré dans les tepms), mais la régie (sont payé au temps de travail et pas à condition de remplir un contrat ). Pareil avec Décath 6 pers font de la régie... Administration système

### 2.1.3 Projets

#### 2.1.3.1 Décathlon

L'un des principaux clients de Néréide est le groupe Decathlon qui se spécialise en grande distribution de produits de sport et de loisirs. Le projet représente une plateforme de vente de puces RFID(bookmark), qui assure l'intégralité du processus d'achat.

pour les achteurs et producteurs, dev front pour le store specifique au lieu d'écrans OFBiz, communication avec des API...

#### 2.1.3.2 Déjbox

Dejbox est une société de la foodTech qui propose aux salariés d'entreprise de leur livrer des repas directement sur leur lieu de travail. L'ensemble des vente est réalisé au travers d'un site e-commerce par lequel le salarié commande un repas.

Le projet a pour objectif de mettre en œuvre un outil de type ERP afin de gérer la chaîne de réapprovisionnement en produit frais vendu en ligne. Il s'agit donc de créer un référentiel d'article et de fournisseur et de pouvoir saisir des commandes qui seront envoyées aux fournisseurs et réceptionnées suite à leur livraison. Enfin, il s'agit de mettre en place la sortie de stock en intégrant les consommations de produit provenant du site de vente en ligne.

.... Dépuis 2013 les dev remontantes à la communauté , parce que besoin de support, divergence. moutons-acteur

## 2.2 Framework OFBiz

### 2.2.1 Vue d'ensemble

*Open For Business (OFBiz)* est une suite d'applications pour la gestion de l'entreprise qui se base sur une architecture très couramment utilisé *(MVC)* et qui implémente des composantes classiques de gestion des donnés, de logique métier, et de traitement spécialisé.

On peut notamment distinguer les modules génériques destinées à la gestion des tâches communes à la plupart des entreprises, telles que la gestion des stocks, la comptabilité, la facturation et bien d'autres. Quant à leur structure, toutes les composantes sont étroitement liées entres elles, ce qui facilite la compréhension, l'utilisation et la personnalisation de ces dernières.

En plus d'une architecture qui encourage la customisation, OFBiz est entièrement distribué en tant que *open source software* [1] ce qui le rend particulièrement intéressant car le logiciel développé à base de OFBiz n'est pas soumis à la condition d'être libre comme c'est le cas de la licence GPL [2] par exemple.

### 2.2.2 Architecture

D'un point de vue purement technique OFBiz se base sur la plateforme Java ainsi que sur l'utilisation des DSL [3] basés sur des grammaires écrites en XML (mettre la bib). En ce qui concerne la partie principale du framework, les échanges HTTP sont implémentes par une extension de la classe `HTTPServlet` [2] et la communication avec les bases de données se fait via l'API Java JDBC [3].

Dans sa structure on distingue *le framework*, *les applications* et *les plugins*. Le *framework* comporte l'ensemble des outils et des mécanismes techniques de bas niveau utilisés par les applications. Il fournit notamment des fonctionnalités présentes dans la plupart des frameworks de développement(couche données, logique métier,gestion des transactions, etc...). Les principaux composants métier tels que la comptabilité, la gestion des stock, ou la facturation se trouvent dans la partie *applications*. Finalement la notion du plugin correspond à une application spécifique qui repose sur des composantes générales : par exemple le plugin *eCommerce* correspond à un boutique en ligne interagit avec des nombreuses *applications* comme *la gestion du stock* ou *la facturation*.

1. Logiciel libre sous licence ASL2 (Apache License Version 2.0) ce qui donne le droit de personnaliser, d'étendre, de restructurer et de vendre le système concerné.
2. GNU General Public Licence
3. Domaine specific language *(Langages spécifiques au domaine)*

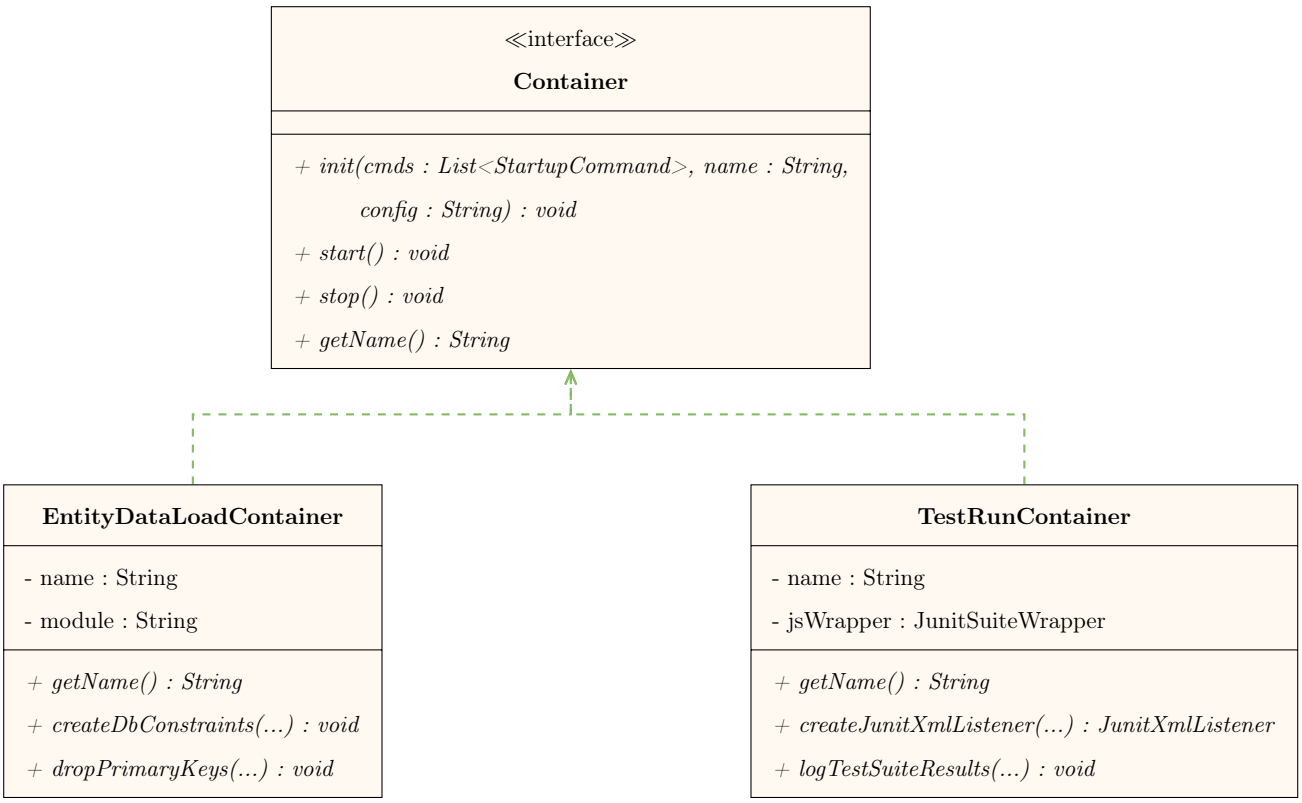



FIGURE 2.1 – Définition du type container

### 2.2.3 DSL XML

L'une des particularités d'OFBiz ce sont des fichiers XML qui servent à déclarer entre autres des routes HTTP, des pages de rendu appelés *Écrans*, ainsi que des services. Le principe est de transformer des informations sous format XML facilement compréhensibles par le développeur, en objets Java correspondants.

### 2.2.4 Container

L'interface container représenté sur la figure 2.1 permet de définir des objets qui correspondent à des processus qui peuvent être initialisés, démarrés et arrêtés. L'intérêt est de pouvoir lancer un daemon spécifique en parallèle de l'execution d'OFBiz comme c'est le cas de `EntityDataLoadContainer` qui est responsable du chargement des données et leur mise à jour en cas de modification du modèle. Quand à `TestRunContainer` il s'assure du lancement des testes unitaires grâce à un mécanisme propre au framework.

### 2.2.5 Composants

Les éléments constitutifs de OFBIZ sont des composants. Un composant est un regroupement des containers, des entités, des services, des vues (*Écrans*) et des applications Web.

L'exemple classique d'un composant est celui de `webtools` qui assure la gestion technique de l'ensemble du système par l'administrateur via une application web, ce qui implique le fait que ce composant regroupe la plupart des éléments majeurs du framework. Nous en

```
<request-map uri="stock">
    <event type="service" invoke="getStock"/>
    <response name="success" type="view" value="stockScreen"/>
</request-map>
```

FIGURE 2.2 – Association d'un point d'entrée et d'une réponse

tant que développeurs avons la possibilité de définir nos propres composants, notamment des *plugins*.

### 2.2.6 Web applications

Des composant OFBiz ne peuvent pas être accédés directement par les utilisateurs, ils servent simplement à organiser le framework en parties individuelles de chaque aspect de l'ERP afin de faciliter leur gestion. Les applications web *(webapps)* sont destinées à fournir un front-end afin que les utilisateurs puissent interagir avec OFBiz. En ce qui concerne les routes HTTP, définis classiquement dans le fichier `web.xml`, dans le cas de OFBiz leur gestion est délégué à un ficher `controller.xml` qui à son tour associe des traitement spécifiques à chaque point d'entrée HTTP ainsi que la valeur de retour qui peut être une vue *(Écran)*, du type `JSON` ou bien une redirection. Cela se fait au moyen d'une `request-map` comme on peut voir sur l'extrait de code suivant 2.2

### 2.2.7 Entity engine

Comme dans beaucoup d'autres frameworks, l'interaction avec les bases de données à une place principale dans le OFBiz. Le moteur d'entités (*Entity engine*) se charge de la communication avec les bases de données à travers les déclarations uniformes, c'est à dire qui changent pas peu importe le choix de l'outil externe de gestion.

### 2.2.8 Service engine

Les services web assurent les échanges d'information être les applications, communément via le protocole `HTTP`. Les services OFBiz fonctionnent dans une architecture orientée service (SOA). Non seulement ces services ont une capacité d'évoquer les autres intérieurement, mais peuvent aussi être appelés par une application extérieure en utilisant des protocoles d'échange d'information tells que `SOAP`.

Les services OFBiz sont appelées en passant un contexte [4] et retournent une réponse parmi celles conventionnellement nommés : *"success"*, comme on peut voir dans 2.2 , *"error"* ou *"failure"* ainsi que l'ensemble des données retournées par le service.

On peut voir l'exemple de la définition d'un service sur 2.3 , qui montre notamment la saisie des attributs attendu par le service qui sont définis de deux manières : En utilisant le mécanisme de `auto-attributes` qui génère des attributs [5] à partir de l'ensemble des clés primaires de l'entité `Stock`. L'autre manière de faire est de rajouter des attributs manuellement comme on peut le voir dans la suite de l'exemple.

4. Définis souvent dans les paramètres de la requête HTTP
5. Qui sont en l'occurence en entrée (de paramètre IN)

```
<service name="getStock" engine="entity-auto" default-entity-name="Stock">
  <auto-attributes include="pk" mode="IN" optional="false"/>
  <attribute name="authKey" type="String" mode="IN" optional="true"/>
  <attribute name="stockList" type="String" mode="OUT" optional="true"/>
</service>
```

FIGURE 2.3 – Définition d'un service

### 2.2.9 Screen engine

La partie Vue du MVC est représentée par des *Écrans* ou les `Screen` qui font partie du `Widget toolkt` [6] de OFBiz. Le principe du fonctionnement d'un écran se base toujours sur un DSL qui interagit avec un mécanisme de rendu générique capable de gérer différents formats de sortie comme `HTML`, `XML`, `CSV`, ou `XLS`. Le cas d'utilisation le plus commun est celui d'une réponse HTML qui est éventuellement générée en déléguant le rendu à un autre mécanisme : `Apache Freemarker`.


FIGURE 2.4 – L'exemple d'un Ecran de rendu OFBiz.

### 2.2.10 Fonctionnel métier

Le méta-composant *Applications* comporte des composants métier prédéfinis, qui ont pour vocation de fournir des solutions fonctionnelles "prêtes à l'emploi".

Ainsi on distingue parmi d'autres :

**accounting** fournit l'ensemble de services et d'écrans de rendu pour la comptabilité.

**datamodel** s'occupe de la définition des entités et des relations entre elles.

**humanres** correspond à la gestion du personnel.

**product** permet la gestion du catalogue des produits et des services fournis par l'organisme.

6. Boite d'outil de composant d'interface graphique

## 2.3 Sujet de stage

Comme mentionné avant, le sujet de stage est divisé en deux parties. La première partie consiste à prendre en main l'outil de développement concerné, le framework OFBiz. La deuxième partie du stage est la modification de certains éléments du framework, notamment des mécanismes de gestion de services afin d'assurer la conformité au style architectural REST.

### 2.3.1 Découverte OFBiz

Cette étape sert à se familiariser avec l'environnement du framework à travers des tutoriels et l'analyse des projets existants, afin de comprendre le fonctionnement général. Cela permet aussi de repérer les points essentielles auxquelles il faut tenir compte lors de la deuxième phase de travail.

### 2.3.2 API REST au sein d'OFBiz

Finalement, l'intérêt principal du stage est la mise en place d'un système de gestion des services REST. L'idée a été évoquée pour la première fois dans une discussion communautaire Apache OFBiz, car le mécanisme en cours nécessitait de l'évolution. Cette discussion a suscité des nombreuses remarques en terme de faisabilité et a permis de retrouver des pistes à poursuivre lors de l'implémentation.

Dans un premier temps on considère la possibilité d'intégration d'une solution externe notamment à travers des librairies JAX-RS de CXF, ainsi que Apache Camel. Malgré les premières prototypes fonctionnelles l'idée d'une solution externe a été abandonné pour des raisons expliqués plus loin dans le document. A sa place une implémentation de bas niveau a été adoptée. Une fois le système mis en route, l'étape suivante consistait à faire adopter la modification dans la branche principe du projet OFBiz.

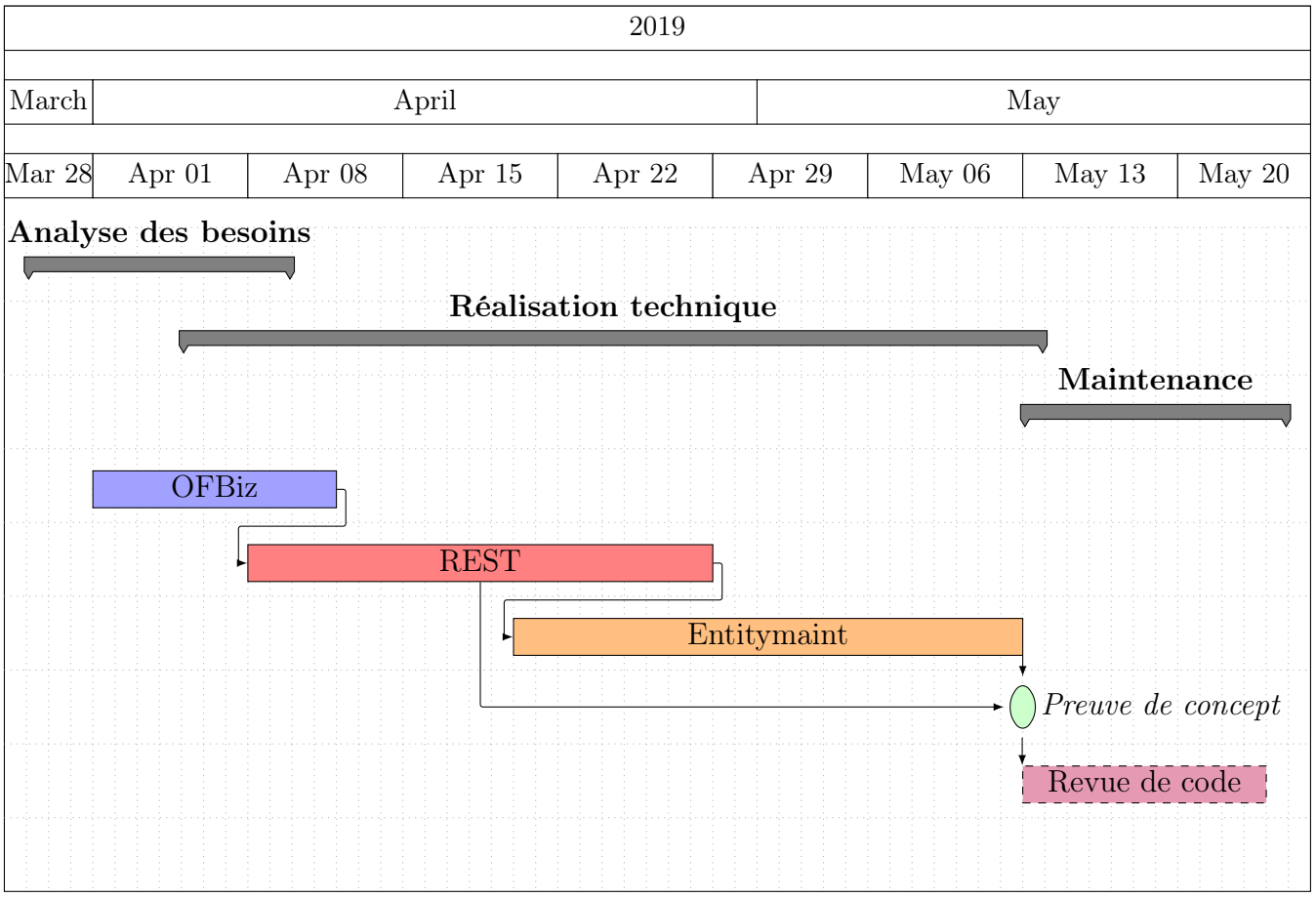
Afin de prouver la conformité du nouveau système et démontrer les nouveaux fonctionnalités, la décision a été prise de modifier une partie front-end existante, notamment l'interface de gestion des entités *entitymaint*. Finalement, l'ensemble de modification sous la forme des patchs a été soumis à la communauté afin d'être revus et potentiellement adoptés dans le framework.

# Chapitre 3

# Travail réalisé

## 3.1 Aperçu général

Voici la chronologie du travail réalisé en entreprise.



## 3.2 Environnement

### 3.2.1 Installation de l'environnement

Avant tout, mon intégration a commencé par l'installation du poste de travail suivi par une discussion sur le choix de distribution Linux , la configuration des outils utilisés par l'entreprise ainsi que par la mise en place des accès aux ressources internes. Le choix d'IDE à été fait au faveur de IntelliJ car il possède des nombreux moyens de navigation qui sont incontournables dans la structure de OFBiz riche en dualité XML/Java.

### 3.2.2 Formation générale

Lors de la formation générale, traditionnellement prévue pour tout les nouveau venus de Néréide, j'ai pu découvrir le fonctionnement basique de OFBiz à travers les démonstration des projets existants. Les points soulevés comportait la gestion des dépendances à travers `Gradle` et `Ant` [1], une introduction au langage Groovy et les raisons pour lesquelles il a été préféré au DSL propre à OFBiz [2]

### 3.2.3 Jira

Un autre point intéressant était le système de gestion de tickets `Jira` utilisé par l'entreprise qui permet de suivre et gérer les bugs tout en interagissant avec les clients. L'outil est utilisé également par la plupart des projet communautaires Apache, dont OFBiz.

### 3.2.4 Approfondissement de Git

Même si la gestion de versions de OFBiz était historiquement géré par l'outil Apache SVN, dans la gestion de ses propres projets, l'entreprise a fait le choix pour un système plus moderne - Git.

Alors que j'avais déjà une certaine maîtrise basique de Git, je n'ai jamais eu l'occasion de travailler dans un projet qui comporte des dizaines de branches qui évoluent quotidiennement. Donc, pour monter en compétences sur ce point-là j'ai utilisé le site d'apprentissage conseillé par mon maître de stage : Learn Git Branching

### 3.2.5 Découverte de communauté libre Apache

OFbiz est un projet libre, maintenu par les intéresses qu'on peut catégoriser comme :

**Contributeurs** ceux qui suggèrent des modifications utiles au projets mais ne modifient pas la branche principale.

**Commiteurs** sont des contributeurs responsables de la validation des modifications du framework ainsi que de leurs adoption dans le code source.

**Membres de PMC** sont responsables des décision sur la structure générale du projet et sur la cohérence des modification vis-à-vis de cette dernière [3]

1. Ant étant déprécié depuis la version 16.11 de OFBiz mais certains projets client l'utilisent toujours car ils se basent sur une version antérieure.
2. Pour remplacer le DSL en XML sous le nom Mini lang, en train d'être entièrement déprécié.
3. PMC acronyme de Project Management Committee (Comité de gestion du projet)

## 3.3 Prise en main d'OFBiz

### 3.3.1 Premier plugin

### 3.3.2 Projets existants et leur structure

#### 3.3.2.1 Décathlon

RFID et tout ça

#### 3.3.2.2 Dejbox

Pierre et Antoine ont tout géré

### 3.3.3 Problématique vis-à-vis du développement

What is "fonctionnel", La prob

## 3.4 Etat de l'art

### 3.4.1 Histoire et problématique des applications web

Avec l'évolution des technologies du réseau, on a obtenu à la fin des annnées 60 - début des années 70, la possibilité d'échanger des information numériques être les machines. Cela a permis l'émergence des systèmes d'échange d'information de plus en plus efficaces. Au début, il s'agissait des architectures très simples, avec une seule couche où une machine unique (*le serveur MainFrame*) effectuait tous les traitement relatives et qui était accédé par un terminal passif [4]. Cela présentais l'avantage d'un système centralisé et homogène facile à implémenter, mais des nombreux inconvénients comme la complexité de maintenance du code monolithique et panne générale en cas d'indisponibilité du MainFrame.
L'arrivé des ordinateurs personnelles a permis la séparation de la couché présentation et parfois aussi de la couche application qui étaient désormais placés sur la machine des clients. Grâce à cela on a pu proposer des affichages plus sophistiqués (des clients lourds et légers avancées). C'est notamment avec l'arrivé des architectures à 2-tiers que a fait l'apparition la notion de service - une fonctionnalité fournie par le serveur.

La manière d'évoquer ces services était définie par une interface standardisée, d'où la notion de API (Application Programming Interface), ou bien interface de programmation applicative qui sert à proposer des points d'entrée pour un nombre de clients de nature différente (utilisateur humain ou une autre machine).
Cependant, l'architecture 2-tiers avait présentait aussi des inconvénients majeurs : Si le client interagissait avec plusieurs serveurs il devais comprendre l'API de chacun d'entre eux. De plus, c'était dans la responsabilité du client de gérer l'ordre des appelles, la cohérence et la combinaison des données reçues. S'ajoute à cela les serveurs qui pouvais pas communiquer l'un avec l'autre, résultat - la complexité grandissante des clients.

L'arrivé d'une architecture à 3 tiers (voire N-tiers avec les couches sécurité, couche routage, etc...) et ainsi que de la notion de Middleware [5] ont apporté un certain nombre d'avantages en terme de réduction de complexité coté client et d'interopérabilité entre ces différentes couches. De plus, de nombreux standards appelées WS-* ont été adoptés afin de combler les failles apparues suite à l'utilisation répandue de Middleware. Suite à cela des dévélopeurs et des sociétés ont commencé à mettre en oeuvre tout le stack WS-* même pour des taches où l'utilisation du HTTP suffirait. Concrètement, le HTTP était réduit au protocole de transport avec une énorme charge XML transmise qui définissait l'échange.
Cette approche fonctionne relativement bien dans le contexte d'un seul ou plusieurs organismes qui partagent le même systeme d'information. Mais dès il s'agit de proposer des services à l'exterieur (à l'échelle mondiale via Internet), cela s'avèrait extêmemment compliqué.

### 3.4.2 Representational state transfer

La déviation du protocole HTTP

4. Le terme français pour l'équivalent anglais moins gentil, *Dumb terminal* - une machine sans capacités de calcul qui sert uniquement à afficher l'information reçue

5. Des intergiciels destinés à lier des systèmes informatique de nature différente

#### 3.4.2.1 Histoire

Roy Fielding

#### 3.4.2.2 Principe

*Détailles du cours de Maurizio : idempotence, navigabilité par hyperlink, notion de ressource etc.

#### 3.4.2.3 Avantages

#### 3.4.2.4 Examples d'API du style REST

API REST de Twitter, SoundCloud, Wiktionnaire,
les différences entre la définition de Roy Fielding et l'implémentation de ces dernières

### 3.4.3 Implementations existantes

#### 3.4.3.1 Camel

#### 3.4.3.2 JAX-RS

Tentative d'intégration —
ServletJaxRS fonctionnelle
Particularités techniques (annotations)
Conflit politique car n'est pas dans le même esprit de l'existant.

## 3.5 Analyse de l'existant

Lors de la découverte de OFBiz j'ai prêté particulièrement attention à la gestion des échanges web afin de soulever les points importants relatives à l'implémentation des services REST.

### 3.5.1 Gestion des application web dans OFBiz

### 3.5.2 ControlServlet

### 3.5.3 Controleur

<request-map>...

### 3.5.4 API en cours

RPC

### 3.5.5 Mécanisme de résolution des URI

### 3.5.6 *OverrideView()* et le conflit avec les URI segmentées

### 3.5.7 Filtres

Delegateur et Dispatcher

## 3.6 Analyse des besoins et attentes de la maîtrise d'ouvrage

### 3.6.1 Besoins d'évolution

Avenir *Discussion communautaire*

## 3.7 Réalisations techniques

### 3.7.1 Librairie CXF

Problèmatique avec les dépendances supplementaires : Tika contient déjà le CXF

### 3.7.2 Choix vers URITemplate

description de classe

### 3.7.3 *OverrideView()* et le conflit avec les URI segmentées

### 3.7.4 Choix d'intégration en parallèle avec le système existant

### 3.7.5 Nouveau contrôleur

#### 3.7.5.1 Compromis pour les conflits d'URI

### 3.7.6 Modification de la partie "Administration : gestion des entités" (entitymaint)

#### 3.7.6.1 Choix de la partie illustrative

#### 3.7.6.2 PUT vs POST

#### 3.7.6.3 Clés composées

#### 3.7.6.4 Formulaires génériques

Create update dans un même formulaire.

### 3.7.7 Stateless

#### 3.7.7.1 Les réalisation par la communauté

Jaques Le Roux Token en gardant la session.

### 3.7.8 RESTClient pour la communauté

#### 3.7.8.1 Généralisation de code

#### 3.7.8.2 Correction d'incohérences

# Chapitre 4

# Conclusion

## 4.1 Lien avec les connaissences obtenu lors de la formation universitaire

METTRE DANS LA CONCLUSION

### 4.1.1 MVC

### 4.1.2 Servlet

### 4.1.3 FreeMarker -JSP

### 4.1.4 Notion d'entié - Symfony

### 4.1.5 Routage

### 4.1.6 Gradle Maven

# Bibliographie


[1] Auteur Ailleurs. Titre3. <http://www.url2.org/>, 2014. [Online ; accessed 16-January-2014].

[2] authoe. Jdbc. https ://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/, date.

[3] authoe. Jdbc. https ://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/, date.

[4] Auteur Autre. Titre2. <http://www.url1.org/>, 2014. [Online ; accessed 16-January-2014].

[5] Auteur Elle. Titre5. <http://www.url4.org/>, 2014. [Online ; accessed 16-January-2014].

[6] groo. groovy.

[7] Auteur Livre1. *Titre Livre1*. Editeur1, 2014.

[8] Auteur Lui. Titre4. <http://www.url3.org/>, 2014. [Online ; accessed 16-January-2014].

[9] Auteur Untel. Titre1. <http://www.url0.org/>, 2014. [Online ; accessed 16-January-2014].