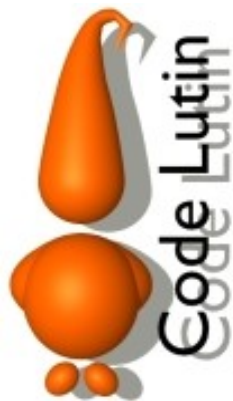


Florian Desbois



# Chorem

Développement d'une architecture logicielle pour applications de gestion d'entreprise



Maître de stage : Benjamin Poussin

Tuteur de stage : Denis Béchet

# *Remerciements*

Benjamin Poussin, dirigeant Lutin, pour son encadrement durant le stage.

Arnaud Thimel et Eric Chatellier, développeurs Lutins, pour leurs aides à l'utilisation de ToPIA.

Stéphane Chorlet, développeur Lutin, et Nolwenn Rannou, stagiaire Lutin, pour leurs expériences de Tapestry.

Julien Ruchaud, développeur Lutin, pour ses explications sur le Chorex existant et sa bonne humeur communicative.

Sylvain Letellier, développeur Lutin, pour son soutien et son aide dans l'utilisation de JRst.

Tony Chemit, développeur Lutin, pour son aide à la création des projets avec Maven et Subversion et son petit grain de folie.

Denis Béchet, tuteur de stage à l'université, pour le suivi de mon avancement durant toute la durée du stage.

Je remercie également tous les autres lutins non cités ici, Estelle Gendron, Yannick Martel et Jean Couteau, pour leurs bonne humeur et leurs soutiens pendant le stage.

Je remercie également les professeurs de l'université de Nantes et du Master 2 ALMA, en particulier le responsable Gerson Sunyé, qui m'ont permis de terminer mes études dans une bonne ambiance avec une très bonne formation en terme d'architecture logicielle.

# TABLE DES MATIERES

<b>Introduction.....</b>	<b>4</b>
<b>I- Contexte du stage.....</b>	<b>5</b>
1 - <i>Présentation de l'entreprise.....</i>	5
2 - <i>Étude de l'existant.....</i>	6
3 - <i>Objectif et problématique.....</i>	7
4 - <i>Démarche.....</i>	8
<b>II- Architecture.....</b>	<b>9</b>
1 - <i>Environnement de développement .....</i>	9
2 - <i>Recherche .....</i>	10
3 - <i>ChoReg (réponse à la problématique) .....</i>	12
a) <i>Objectif .....</i>	12
b) <i>Architecture.....</i>	12
b) <i>Conception.....</i>	14
c) <i>Utilisation.....</i>	16
<b>III – Applications.....</b>	<b>19</b>
1 - <i>Groupe d'applications Chorem .....</i>	19
a) <i>Interactions.....</i>	19
b) <i>Bonzoms, Billy et Ca\$h .....</i>	21
2 - <i>Framework ToPIA .....</i>	22
a) <i>Présentation .....</i>	22
b) <i>Exemple de Bonzoms-business.....</i>	24
c) <i>Contributions .....</i>	25
d) <i>Export des données dans ChoReg.....</i>	26
3 - <i>Framework Tapestry .....</i>	27
a) <i>Présentation .....</i>	27
b) <i>Exemple de Bonzoms-ui (IHM) .....</i>	29
<b>IV – Gestion de projet .....</b>	<b>34</b>
1 – <i>Organisation du projet.....</i>	34
2 - <i>Problèmes rencontrés .....</i>	34
3– <i>Planning réel.....</i>	35
<b>Conclusion .....</b>	<b>36</b>
<b>Bilan personnel.....</b>	<b>37</b>
<b>Bibliographie.....</b>	<b>38</b>

## Introduction

---

Les entreprises, quelque soit leurs domaines d'activités ont toujours le même besoin en terme de logiciel de gestion. Que ce soit pour gérer les personnes de la société, leurs contacts, la comptabilité, leurs projets et les factures de leurs clients. Chorem est un progiciel de gestion intégré (PGI) simple, voulant répondre à la gestion quotidienne des sociétés, tout particulièrement celle de Code Lutin. Il dispose de plusieurs modules répondant chacun à des besoins particuliers : la gestion de projet, la gestion des contacts ou encore la comptabilité.

Ce rapport décrit le stage effectuée sur Chorem dans la société Code Lutin du 1er avril au 28 août 2009. Ce stage se situe dans le cadre de la formation du Master 2 ALMA à l'université de Nantes.

Ce stage est particulièrement orienté architecture logicielle avec un environnement Java. De multiples technologies ont été utilisées pour permettre le développement des différentes applications réalisés au cours du stage.

Ce rapport se décompose en quatre parties :

- **Contexte du stage**, comprenant la présentation de l'entreprise, la description du sujet et sa problématique.
- **Architecture**, comprenant la description de l'architecture logicielle conçu pour le projet.
- **Applications**, comprenant la description des différentes interactions entres les applications qui rentrent dans le cadre du projet, ainsi que la description des deux principaux framework utilisés, ToPIA et Tapestry en se basant sur les exemples concrets des applications réalisés pendant le stage.
- **Gestion de projet**, comprenant l'organisation du projet, les problèmes rencontrés et le planning réel du stage.

## I- Contexte du stage

---

### 1 - Présentation de l'entreprise

Code Lutin est une SSLL (Société de Service en Logiciel Libre) créée en 2002 implantée près de Nantes, à Saint-Sébastien-sur-Loire. Cette société composée d'une dizaine de personnes, est spécialisée dans les technologies autour du langage de programmation Java dans le monde du libre. Ainsi les services qu'elle propose vont du développement logiciel, à l'intégration et la maintenance en passant par le conseil et la veille technologique. Code Lutin est spécialisé en JEE et UML notamment pour la MDA (Model Driven Architecture).

Code Lutin a au fil des années travaillé pour différents projets que ce soit pour de grands groupes (Décathlon, Leroy Merlin, Alcatel, ...), pour des laboratoires de recherche (Ifremer, Cemagref, ...), ou pour des établissements universitaires (Ecole des Mines, Université de Nantes, ...). Comme l'atteste leur présentation web :

*« Code Lutin a construit au fil de l'expérience acquise un processus propre hautement outillé avec des objectifs tels que l'obtention d'un produit fidèle au besoin exprimé par le client, l'amélioration constante de la qualité du logiciel, l'assurance de conformité et de non régression grâce aux tests unitaires, le respect des délais. »*

Code Lutin est également un acteur très impliqué dans la communauté libre. Adhérent à Linux-Nantes depuis 2002 et membre fondateur du Réseau Libre-entreprise la même année et d'Alliance Libre en 2007, Code Lutin ne cesse de montrer son implication à la philosophie libre qu'il soutient, notamment à travers les Rencontres Mondiales des Logiciels Libres (RMLL) organisé récemment à Nantes (du 7 au 11 juillet 2009). Cette organisation en réseau permet à Code Lutin de proposer une large offre de services et de produits autour du logiciel libre et de disposer, au-delà de ses ressources propres, d'expertises pointues dans d'autres domaines.



## ***2 - Étude de l'existant***

Chorem existe depuis quelques années et a évolué sous plusieurs versions avec différentes technologies. Une version actuellement utilisée fonctionne avec le framework web Mentawai et a été réalisée par Julien Ruchaud, développeur chez Code Lutin. Les besoins applicatifs ont évolué notamment dans l'optique d'ajouter de nouveaux modules comme un calendrier partagé ou une gestion plus complète des personnes et des sociétés. Chorem a donc été repris par une équipe d'étudiants pour rendre plus performant le modèle et utiliser un autre framework web plus flexible : Tapestry. Cependant la mise en place de la nouvelle version de Chorem n'a pas été aboutie et nécessite une relecture complète.

Chorem est depuis devenu le nom de l'ensemble des applications liés à la gestion d'entreprise réalisés par Code Lutin. Ainsi d'autres applications déjà existantes ont rejoint le groupe :

- Pollen : application de vote
- JTimer : gestion du temps réel sur des tâches et projets
- Callao/Lima : gestion comptable

Un site internet, et plus particulièrement une forge applicative (listes de diffusions, dépôts svn, bug tracking, ...), a été créée pour regrouper ces applications : [chorem.org](http://chorem.org). L'ancien projet Chorem n'en fait plus partie et doit être entièrement refait en décomposant chaque partie en application indépendante.

### **3 - Objectif et problématique**

Le but du stage est de trouver une architecture permettant de simplement faire communiquer l'ensemble des applications Chorem tout en gardant leurs indépendances. D'autre part, les différents modules de l'ancienne version de Chorem doivent être réécrits dans différentes applications qui s'intégreront à la plate-forme Chorem.

L'existant se traduit en cinq applications à réaliser en partant de la spécification détaillée du Data Model Resource Book (A Library of Universal Data Models for All Enterprises) de Len Silverston :

- Gestion des personnes, sociétés, contacts
- Gestion de projet
- Gestion des ressources humaines (salaires, contrats, ...)
- Gestion de la trésorerie prévisionnel
- Gestion des factures

Chacune des applications doit utiliser le framework ToPIA pour la gestion de la persistance des données et le framework Tapestry pour l'interface utilisateur web. Chacune des applications doit également être indépendantes et pouvoir fonctionner sans aucune contrainte fonctionnelle des autres applications.

**Problématique** : Comment permettre la communication entre toutes les applications Chorem tout en gardant leurs indépendances ? et quelle technologie choisir ?

## 4 - Démarche

La première étape du stage est d'effectuer des recherches sur différentes architectures logicielles existantes répondant à la problématique. Notamment trouver la bonne technologie pour répondre aux contraintes de modularités.

En parallèle, l'étude des spécifications du Data Model ressource Book doit être faite pour réaliser l'analyse et la conception de la première application, celle pour la gestion des personnes, sociétés et contacts baptisée **Bonzoms**.

La deuxième étape consiste à mettre en œuvre un essai d'implémentation avec l'architecture trouvée répondant aux contraintes de modularité. Pour cela un autre projet utilisant les données du premier doit être analysé et conçu.

Une fois les essais concluants sur deux projets, un troisième doit être mis en place pour permettre de tester la robustesse de l'architecture sur trois différentes applications qui interagissent entre elles mais, fonctionnent indépendamment les unes des autres si besoin est.

Nous allons d'abord décrire l'architecture logicielle réalisée pour répondre aux contraintes de modularité. Ensuite nous décrirons les différentes interactions possibles entre les applications Chorem puis la description de celles réalisées au cours du stage tout particulièrement via le framework permettant la génération de code, ToPIA. Dans un dernier temps nous verrons les IHM des applications et l'utilisation du framework Tapestry.

## II- Architecture

---

### 1 - Environnement de développement

Le développement des applications Chorem a été fait via l'IDE (Integrated Development Environment) **NetBeans**. Ceci afin d'intégrer facilement les deux principaux outils utilisés pour le développement : Maven et Subversion.

**Maven** permet de construire les projets. Il gère les dépendances des bibliothèques externes ainsi que le cycle de vie du projet : compilation, génération du code, génération de la documentation, création de l'archive (jar, war, ...) et déploiement. Maven permet ainsi de simplifier toutes les étapes de construction du projet en utilisant différents plugins paramétrables (notamment pour la génération de code).

**Subversion** permet quant à lui de gérer les versions des fichiers sources du projet. Ainsi chaque modification pourra être sauvegarder, permettant un historique des fichiers et un retour arrière possible.

Les projets utilisent également une forge applicative sur Internet, **Redmine**, qui permet aux utilisateurs de suivre l'avancement du projet, de s'abonner à des listes de diffusion du projet, de voir les dernières modifications via Subversion (SVN) ou d'ajouter de nouvelles demandes d'évolutions ou de patches correctifs pour les éventuelles erreurs.

L'une des contraintes majeure de l'architecture est l'indépendance des projets. Ainsi chaque projet à son propre **pom** maven (fichier pour la construction du projet), et son propre SVN. Les dépendances externes sont gérés par Maven sur l'ensemble des bibliothèques et technologies utilisés (ToPIA, Tapestry, Servlet, XML, ...). Chaque projet peut donc être construit et déployé indépendamment dans un serveur d'application comme **Tomcat**, Jetty ou JBoss.

Pour l'instant aucune contrainte de distribution n'est à prendre en compte. Il faut cependant garder à l'esprit que l'architecture doit être flexible pour permettre un déploiement sur plusieurs serveurs distants. Mais la première réflexion consiste à rendre stable une architecture déployée sur une seule machine dans le même serveur d'application.

## 2 - Recherche

Lors des recherches sur l'architecture, une technologie prometteuse fût testée pour répondre aux contraintes du projet : **OSGi**. Son framework à base de composants dynamiques permet une gestion précise des interactions entre chacun d'eux tout en permettant une utilisation « à chaud » sans besoin de recharger le serveur.

Chaque fichier **jar** représentant une application Java correspond à un **bundle** dans l'environnement OSGi. Chaque bundle est décrit précisément par un fichier de manifest comprenant les services fournis par le bundle, les services requis (obligatoires ou non) et une identification précise du bundle (nom et version). Chaque bundle peut également gérer les services requis lorsque l'application est démarrée et s'adapter en fonction des autres bundles également démarrés au sein du même serveur.

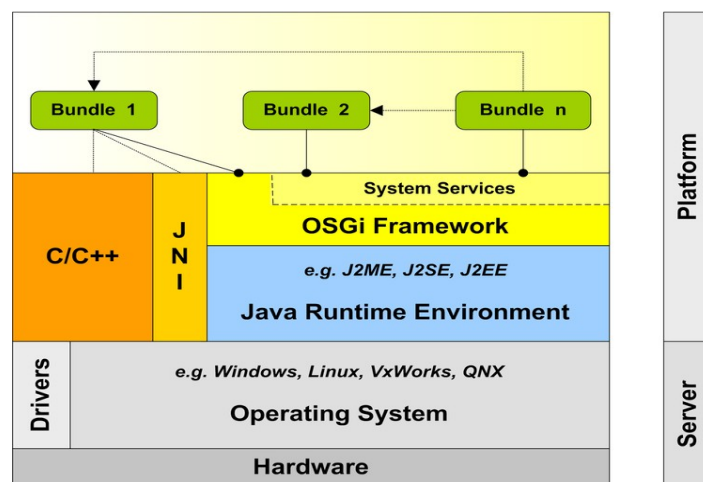


Illustration 1: Architecture OSGi

La contrainte d'indépendance est donc respectée. Les applications peuvent également être démarrées et arrêtées « à chaud » et elles peuvent communiquer suivant leurs présences au sein du même serveur. Cependant il semble peu probable de pouvoir étendre ce principe sur une architecture distribuée. En effet, les bundles doivent être déployés sur le même serveur pour pouvoir utiliser les fonctionnalités intéressantes d'OSGi, sinon cela revient à une simple architecture SOA (Service Oriented Architecture) sans réel plus-value apporté par la technologie.

Les essais ont été effectués via l'implémentation d'OSGi faites par Apache : le framework Apache Felix. Quelques problèmes sont survenus lors de l'utilisation des bundles propres à Hibernate et à la base de donnée H2 (un problème de communication lié à l'indépendance des ClassLoader de chaque bundle). La solution OSGi n'a donc pas été travaillée plus en profondeur et n'a donc pas été retenue comme base pour l'architecture du projet.

D'autres solutions architecturales ont été envisagés, notamment l'utilisation des messages avec JMS (Java Message Service) et/ou MOM (Message Oriented Middleware). L'idée étant de permettre à chaque application Chorem d'envoyer des messages qui seront interprétés ou non par d'autres applications susceptibles d'utiliser les même données. Cette idée n'a pas été totalement écartée et sera potentiellement utilisable dans l'architecture définie.

L'architecture définie ici pour les besoins du projet est une architecture SOA permettant l'enregistrement et le contrôle des services en cours d'exécution. Un registre se charge de centraliser les demandes des applications pour chaque type de données désirés. Ce registre a été intégré à un nouveau projet qui sera le cœur de l'architecture : le projet **ChoReg**.

### 3 - ChoReg (réponse à la problématique)

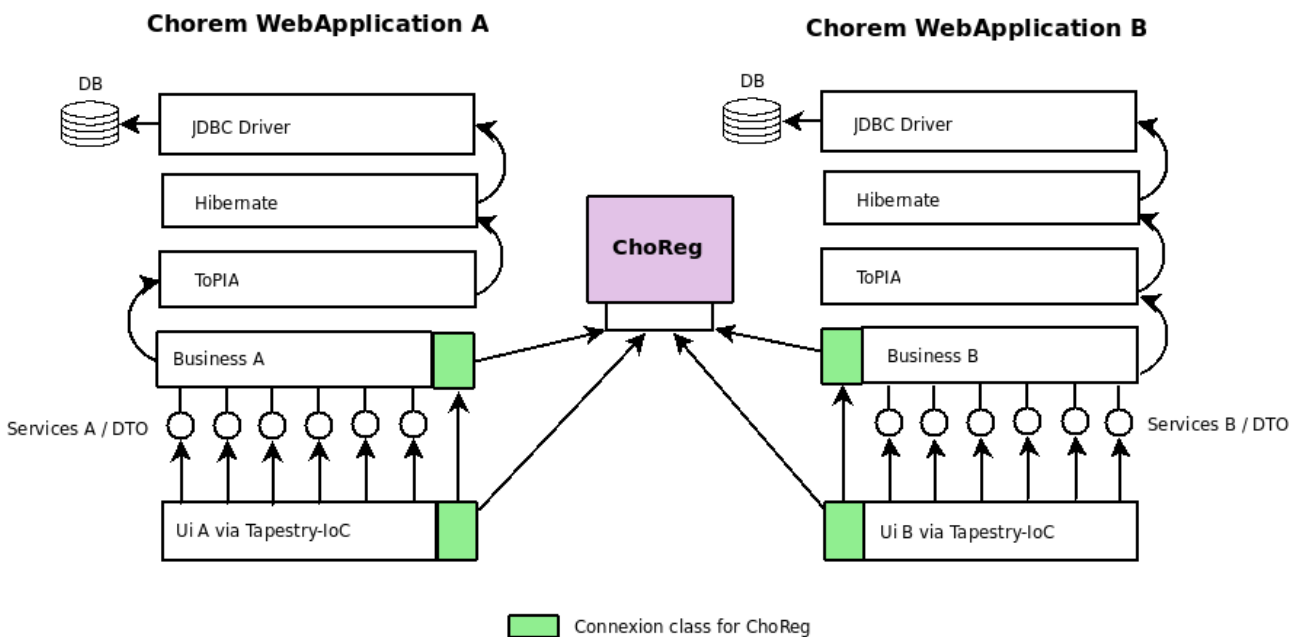
#### a) Objectif

ChoReg ou Chorem Registry permet l'enregistrement de services susceptibles de proposer des données suivant un format standard (norme). Son objectif au sein du groupe d'applications Chorem est de permettre la communication entre les différentes applications. Ces applications ne savent pas avec qui elles communiquent (excepté ChoReg), elles connaissent uniquement les données qu'elles manipulent et souhaitent récupérer via ChoReg. Ainsi les échanges se font suivant des normes (de préférence libre) afin de permettre l'interopérabilité avec d'autres applications de la communauté libre qui supporteraient la même norme. Chaque application Chorem reste indépendante et peut exporter ses données suivant une ou plusieurs normes / formats standards pour enrichir les fonctionnalités d'une autre application. Il est également possible de rajouter une couche de sécurité pour filtrer les demandes suivant les droits d'utilisation spécifiques aux applications.

#### b) Architecture

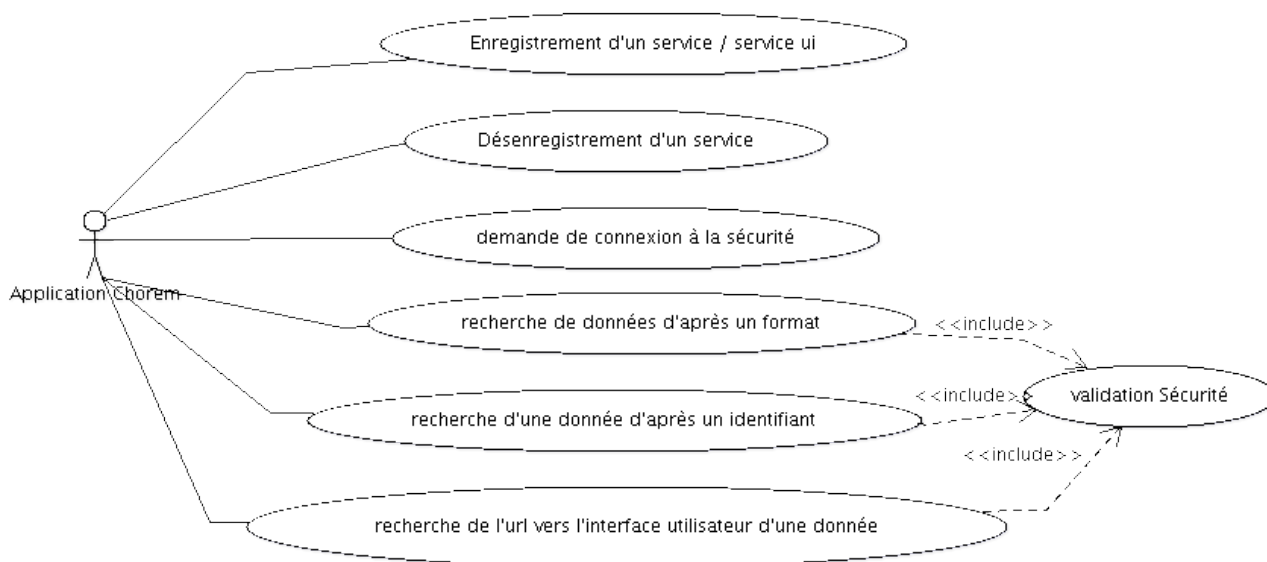
L'architecture des applications Chorem suit un modèle en couche. Chaque application possède deux modules distincts :

- **module business** : Couche métier de l'application basé sur le framework *ToPIA*. Cette couche fournit un ensemble de services qui seront utilisés par l'interface utilisateur web de l'application. Ces services pourront être enregistrés dans ChoReg pour fournir des données à d'autres applications.
- **module ui** : Interface utilisateur web de l'application basé sur le framework *Tapestry*.



ChoReg devra répondre à la demande de chacun des modules : Un service métier pourra être enregistré pour la manipulation des données et un service ui pourra être enregistré pour la manipulation des interfaces utilisateurs web et plus particulièrement de leurs urls.

Voici donc les cas d'utilisations de ChoReg :



*Illustration 2: Diagramme de Cas d'Utilisations*

Note : Nous décrivons dans la 3ème partie du rapport, un exemple d'application et le détails de ses deux modules « business » et « ui » qui utilisent respectivement les frameworks **ToPIA** et **Tapestry**.

## b) Conception

### Services enregistrés dans ChoReg

Il est possible d'enregistrer trois types de services différents dans ChoReg :

- ChoremService : Service pour la manipulation des données
- ChoremUIService : Service pour la manipulation des urls vers les interfaces utilisateurs
- ChoremSecurity : Service pour la gestion de la sécurité

Il ne peut y avoir qu'un seul ChoremSecurity enregistré dans ChoReg pour simplifier la gestion de la sécurité pour l'accès aux données.

Chaque service enregistré dans ChoReg doit implémenter l'une des trois interfaces précédentes (ChoremService, ChoremUIService ou ChoremSecurity).

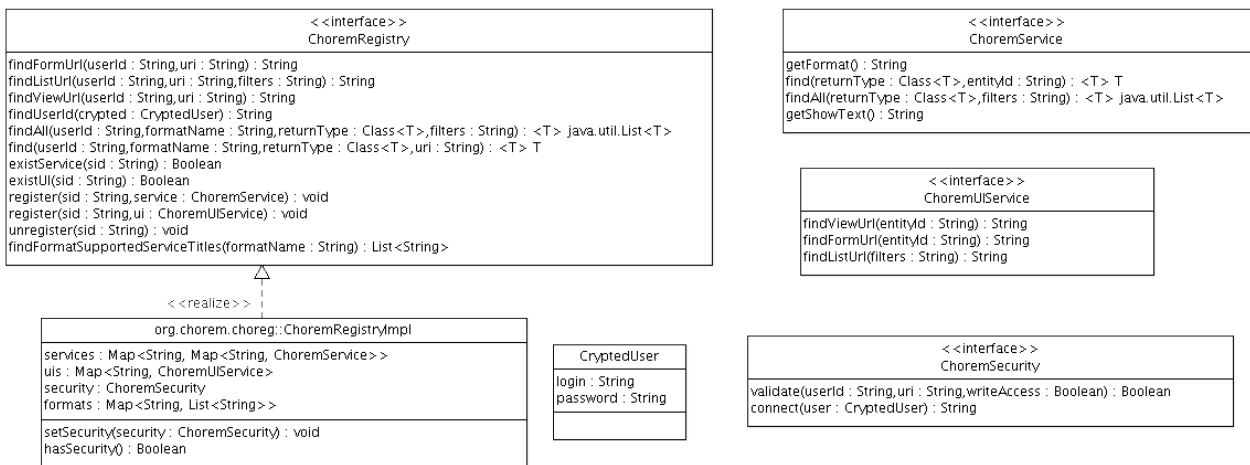


Illustration 3: Diagramme des classes et interfaces de ChoReg

## Identifiants

Chaque service est identifier de façon unique grâce à un identifiant correspondant au nom complet du service appelé sid. Exemple :

`org.chorem.bonzoms.services.ServicePerson`

Les applications Chorem utilisent ToPIA pour la persistance des données. Dans ToPIA, chaque entité (donnée) est identifiée de façon unique grâce au nom complet de l'entité et une chaîne numéroté unique. Exemple :

`org.chorem.bonzoms.persistence.PersonEntity#1246900056944#0.26595214137921097`

Ainsi, ChoReg définit comme étant une choremUri, la concaténation de ses deux identifiants, le sid et le topiald :

`org.chorem.bonzoms.services.ServicePerson://org.chorem.bonzoms.persistence.PersonEntity#1246900056944#0.26595214137921097`

Chaque donnée qui transite par ChoReg doit être identifier de cette manière unique pour être facilement retrouvée.

## **Demandes auprès de ChoReg**

Deux types de demandes peuvent être faites auprès de ChoReg. Suivant un format ou un identifiant (choremUri) pour obtenir :

- soit une liste de données répondant à un format,
- soit la donnée correspondant à un choremUri,
- soit l'url vers l'interface de la liste des données,
- soit l'url vers l'interface du formulaire d'ajout/modification d'une donnée,
- soit l'url vers l'interface de la fiche de détails d'une donnée

La méthode « findAll » de l'interface ChoremService permet la demande de plusieurs données répondant à un format. Le format définit la norme utilisé pour le transfert des données. Généralement il s'agit d'une norme au format XML. Un ChoremService peut utiliser différents formats pour exporter ses données. Un même sid peut donc décrire une ou plusieurs implémentations d'un même service avec un format différent. Ce même sid peut également décrire le service UI (User Interface) lié au ChoremService pour la récupération des urls.

La méthode « find » de l'interface ChoremService permet la demande d'une seule donnée d'après son identifiant unique (choremUri). Le choremUri est également utilisé pour retrouver les urls des interfaces utilisateurs. Un ChoremUIService pourra fournir trois types d'url :

- ListUrl : url vers la liste pour un type d'entité (choremUri = sid://entityType)
- FormUrl : url vers le formulaire d'ajout/modification d'une entité (choremUri = sid://topiald)
- ViewUrl : url vers la fiche de description d'une entité (choremUri = sid://topiald)

### c) Utilisation

ChoReg est utilisé de manière statique dans les applications Chorem. Toutes ces applications manipulent le même singleton de ChoReg. Cela est possible grâce au serveur d'application qui permet l'utilisation de bibliothèques globales à toutes les applications déployées. Le déploiement de ChoReg est effectuée dans le serveur d'application Apache Tomcat 6. Le dossier « lib » du serveur permet de charger les bibliothèques globales au démarrage du serveur. Chaque application ne doit pas avoir la bibliothèque ChoReg dans son fichier « war ». Dans maven, la dépendance sur ChoReg est en « scope » « **provided** » pour que maven n'intègre pas le fichier jar de ChoReg dans le fichier « war » qu'il va créer.

Dans Tomcat, chaque application a son propre ClassLoader<sup>1</sup>. En plus de cela, le dossier « lib » a également son ClassLoader : le ClassLoader « **common** ». Dès qu'une application demande l'existence de ChoReg dans son ClassLoader, elle ne le trouvera pas, et Tomcat ira chercher la bibliothèque dans le ClassLoader common. Si ChoReg n'est pas déployé dans Tomcat, une erreur surviendra dans l'application. Il est important de bien écarter ce cas d'erreur (récupérer l'exception *NoClassDefFoundError*) pour permettre à l'application de fonctionner même sans ChoReg.

Sur le diagramme de déploiement ci-dessous, deux applications web sont déployés, Ca\$h et Bonzoms. La description et les interactions entre ces deux applications seront décrites dans le prochain chapitre de ce rapport.

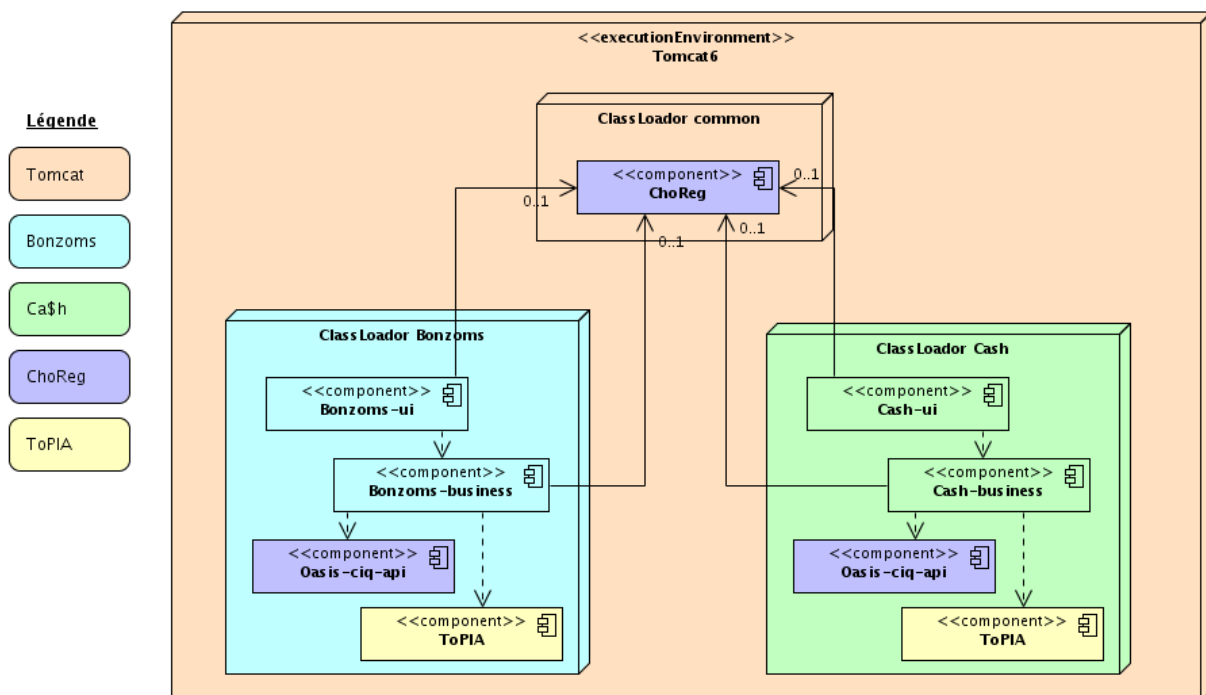


Illustration 4: Diagramme de déploiement dans Tomcat6

Il est important de contrôler le cycle de vie d'une application déployée. Ainsi au

<sup>1</sup> The **Java Classloader** is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine (cf [http://en.wikipedia.org/wiki/Java\\_Classloader](http://en.wikipedia.org/wiki/Java_Classloader) )

démarrage, les services devront être enregistrés dans ChoReg, et à l'arrêt, les services seront désenregistrés de ChoReg. Pour cela, deux méthodes différentes ont été utilisés.

## Démarrage et arrêt d'une application

Nous avons vu qu'une application Chorem possède deux modules, un module « business » pour la partie métier de l'application et un module « ui » pour l'interface utilisateur web. Il est possible d'utiliser le module « business » indépendamment de son interface utilisateur. Ainsi le module « business » doit pouvoir être déployé dans un serveur d'application sous forme d'application web, donc de fichier « war ».

Un fichier war doit posséder un dossier WEB-INF contenant un fichier web.xml décrivant les servlets et autre configuration nécessaire au lancement de l'application. Pour contrôler le lancement et l'arrêt de l'application, une classe est créée et implémente l'interface *ServletContextListener*. Cette interface possède deux méthodes à implémenter :

- **contextInitialized** : lancé au démarrage de l'application web
- **contextDestroyed** : lancé à l'arrêt de l'application web

Pour le cas de l'application Bonzoms, une classe « *RegisterBonzomsServices* » a été créée pour implémenter cette interface et enregistrer les services de l'application qui implémentent l'interface « ChoremService » auprès de ChoReg.

Ainsi le fichier **web.xml** contient les lignes suivantes pour permettre au serveur d'application, en l'occurrence Tomcat, d'utiliser les méthodes implémentées pour contrôler le cycle de vie de l'application :

```
<web-app>
  <display-name>Bonzoms Services</display-name>
  <listener>
    <listener-class>
      org.chorem.bonzoms.RegisterBonzomsServices
    </listener-class>
  </listener>
</web-app>
```

Pour le cas du module « ui » qui est lui même une application web, il faut passer par le framework web utilisé, en l'occurrence **Tapestry**, pour contrôler le démarrage et l'arrêt de l'application. Nous ne décrivons pas ici la procédure d'utilisation de Tapestry pour permettre à l'application d'enregistrer et désenregistrer ses services UI auprès de ChoReg. Le principe reste cependant le même que pour le module « business », une classe implémentant les interfaces pour le démarrage et l'arrêt de Tapestry est utilisée pour enregistrer et désenregistrer les services UI (ChoremUIService) auprès de ChoReg. Cette même classe appelle également les méthodes de *RegisterBonzomsServices* pour enregistrer les services métiers (ChoremService).

Vous trouverez sur la page suivante un exemple d'utilisation de ChoReg : L'application Ca\$h fait une demande de références au format xPIL. Bonzoms a enregistré un service qui supporte ce format dans ChoReg. Ainsi à la demande de Ca\$h, ChoReg ira chercher les données xPIL dans Bonzoms pour les renvoyer à Ca\$h.

# Exemple d'utilisation entre Bonzoms et Ca\$h

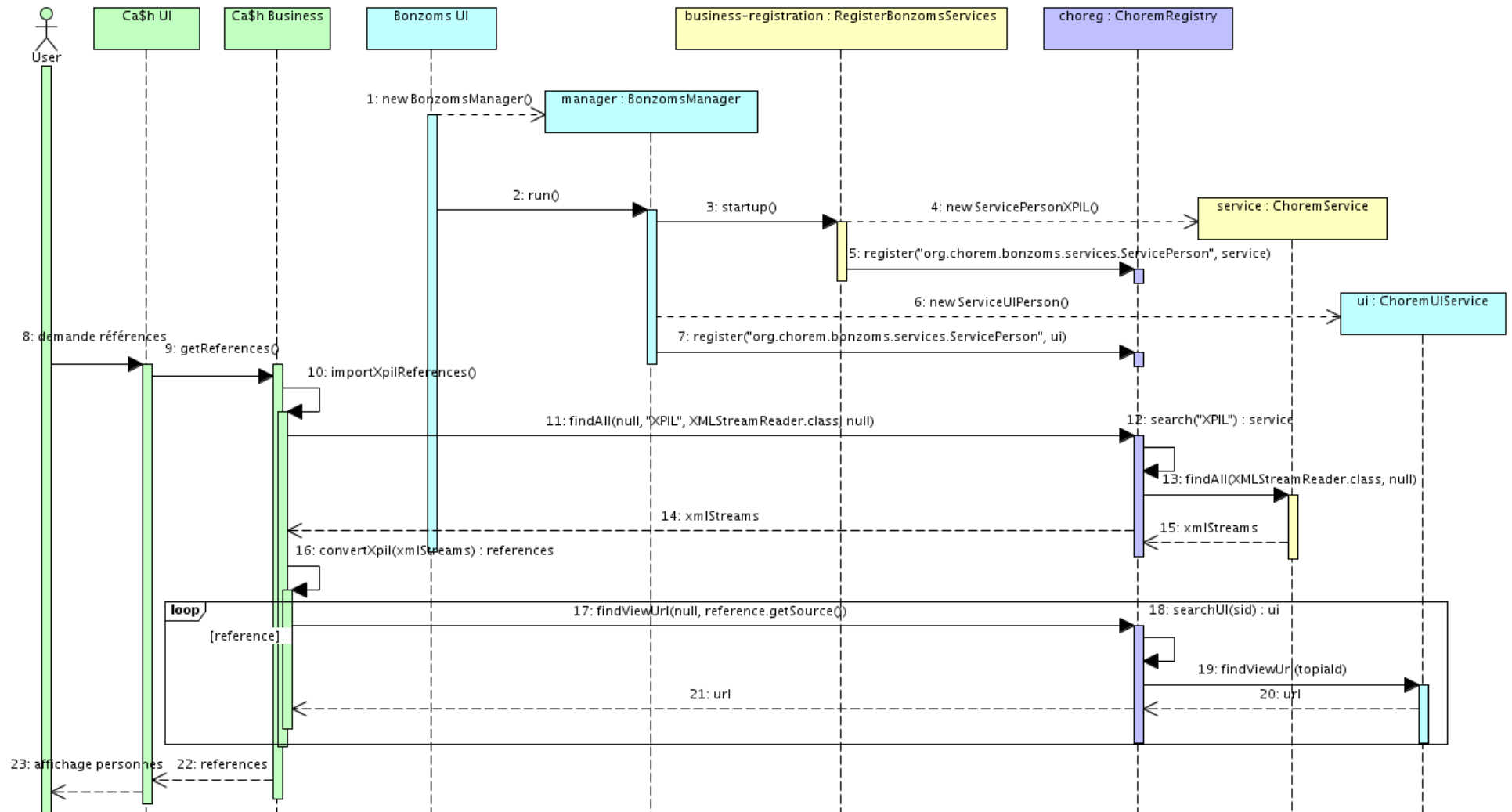


Illustration 5: Diagramme de séquences: Ca\$h fait une demande de références au format xPIL auprès de ChoReg

## III – Applications

---

### 1 - Groupe d'applications Chorem

#### a) Interactions

Au sein du groupe d'applications Chorem existent déjà 3 applications :

- **Callao/Lima** : Gestion de la comptabilité
- **JTimer** : Gestion du temps réels des utilisateurs sur des tâches / projets
- **Pollen** : Application de vote.

A ce groupe se rajoutent trois applications qui ont été créés pendant le stage :

- **Bonzoms** : Gestion des personnes et sociétés, leurs rôles, relations et contacts
- **Ca\$h** : Gestion de la trésorerie prévisionnelle
- **Billy** : Gestion de la facturation

Cinq autres applications n'ont pas encore été réalisés et peuvent être rajoutés au groupe Chorem :

- Gestion des communications (mails, réunions, ...) (non baptisé : **Communication**)
- Gestion d'un calendrier/agenda partagé (**Kalendaro**)
- Gestion de projet (**Gepeto**)
- Gestion des ressources humaines (non baptisé : **HumanResources**)
- Gestion de la sécurité des applications (non baptisé : **SecurityManager**)

Chacune de ses applications peut exporter ses données via ChoReg pour les utiliser dans une autre application.

Par exemple, les liens entre les personnes et sociétés que propose Bonzoms sont utiles pour de nombreuses autres applications :

- Pollen pour permettre la création de groupe de votants par société...
- Kalendaro pour afficher les dates de création des sociétés ou les périodes d'embauche de leurs employés...
- Gepeto pour attribuer des tâches et projets à des personnes...
- HumanResources pour gérer les contrats des personnes envers leurs sociétés...
- SecurityManager pour la gestion des droits d'accès des personnes...
- ...

Ainsi Bonzoms devra répondre aux demandes des différentes applications dans l'export de ses données.

Le diagramme de composants ci-dessous représente les liens possibles entre les applications Chorem. Ces liens sont fictifs car en réalité, toutes les applications passeront par ChoReg pour communiquer.

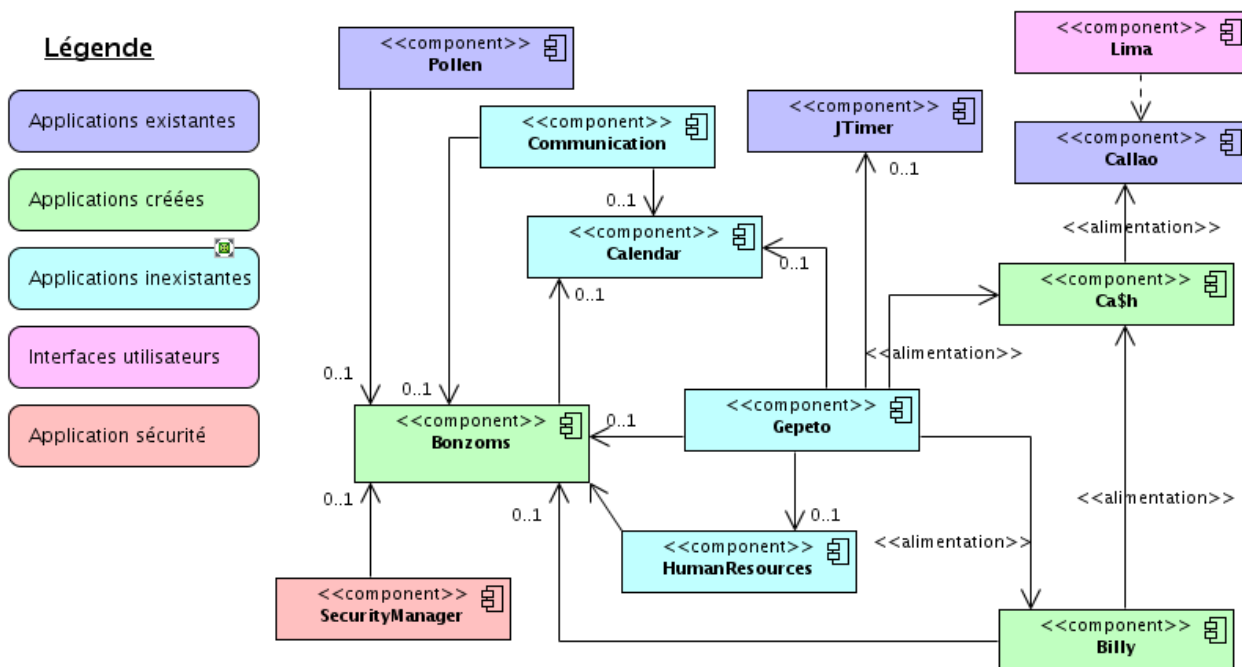


Illustration 6: Diagramme de composants des applications Chorem

Nous allons détailler les relations entre les trois applications réalisées : Bonzoms, Billy et Ca\$h.

## **b) Bonzoms, Billy et Ca\$h**

Bonzoms a été la première application créer. Il a été décider de créer Ca\$h en deuxième pour sa simplicité. Ainsi l'effort sera concentrer sur l'architecture et l'utilisation de ChoReg plutôt que les fonctionnalités de l'application.

Ca\$h est un tableau prévisionnel de la trésorerie de la société qui l'utilise. Cette partie est déjà existante dans l'ancienne application Chorem. Dans sa version existante, le tableau prévisionnel intègre le décompte des factures faites aux clients de la société. Ca\$h n'a donc pas de communication directe avec Bonzoms car cette liaison passe par la gestion des factures et donc par une autre application baptisé Billy. Cette prise de conscience s'est avéré pendant l'implémentation de Ca\$h.

Ca\$h a tout de même été implémenté entièrement, et une page a permis de tester l'utilisation du format d'export des données de Bonzoms via ChoReg. Bonzoms utilise le format xPIL (XML) de la norme Oasis CIQ pour exporter ses données. Ainsi Bonzoms enregistre ses services de manipulations des personnes et sociétés auprès de ChoReg via le format xPIL. Ca\$h a permis de tester la récupération des données en xPIL depuis ChoReg. Depuis, cette partie a été extraite de Ca\$h pour être intégré dans l'application Billy. L'implémentation de Billy n'est pas entièrement terminé, et il n'est pas encore possible d'utiliser ses données (les factures) dans Ca\$h avec l'utilisation d'une autre norme. Le chapitre « Gestion de projet » décrit plus en détails les problèmes d'organisation rencontrés pendant le stage.

Nous allons maintenant détailler les framework utilisés, ToPIA pour la partie « business » et Tapestry pour la partie « ui ». L'application témoin sera Bonzoms.

## 2 - Framework ToPIA

### a) Présentation

**ToPIA** et plus particulièrement son module **ToPIA-persistence**, permet de simplifier les manipulations avec la couche persistante d'une application. Il utilise le framework **Hibernate**<sup>2</sup> pour permettre les communications avec la base de données. **ToPIA** permet également de générer l'ensemble des entités (objets java) qui serviront à l'enregistrement des données en base. Pour se faire il utilise une librairie, **EUGene** permettant de générer n'importe quel diagramme **UML** en objets java simple d'utilisation (**ObjectModel**). Une conversion est faite entre le fichier **XMI** correspondant au schéma **UML** et un fichier **XML** appelé **ObjectModel** permettant une génération simple de la norme **UML** en Java.

Grâce à **EUGene** et **ToPIA**, les classes métiers d'une application représentant les données à enregistrer en base, sont facilement modélisables en **UML** puis générés en objets java. La génération s'effectue à chaque compilation de l'application via Maven. **ToPIA** utilise le pattern **DAO** (Data Access Object) pour permettre de manipuler les entités générés. Chaque entité possède son propre **DAO** pour permettre les manipulations sur l'ensemble de ces instances. Le **DAO** permet, l'ajout, la modification, la suppression d'une entité ainsi que de nombreuses méthodes permettant la sélection d'un ensemble d'entité du même type suivant différents paramètres. Les **DAO** permettent d'abstraire la couche de gestion **Hibernate** permettant la manipulation des entités persistantes en base de données. Plus simplement, les **DAO** permettent d'effectuer de nombreuses requêtes sur la base, tout en utilisant le modèle objet généré. Pour se faire **ToPIA** utilise le principe de transaction. Une transaction globale permet d'initialiser la base de données et le contexte principale de **ToPIA**. A partir de ce dernier, à chaque besoin d'une ou plusieurs requêtes sur la base de données, une ou plusieurs transactions peuvent être créés. Ainsi, respectant les besoins d'une base de données, les transactions peuvent être appliqués (commit) ou annulés (rollback) suivant les besoins du code métier de l'application.

---

2 Hibernate is a powerful, high performance object/relational persistence and query service. Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. (cf [www.hibernate.org](http://www.hibernate.org))

**ToPIA** génère pour chaque entité du modèle (chaque class UML avec le stéréotype « entity »), 7 fichiers : 3 pour l'entité, 3 pour son **DAO** et un dernier fichier permettant le mapping des attributs et classes pour l'utilisation d'Hibernate.

Sur le diagramme ci-dessous est représentée la hiérarchie d'héritage des classes générées avec celles de **ToPIA**. Ainsi seront manipulés principalement les objets DAO (le plus bas de la hiérarchie) considéré comme un **TopiaDAO** (interface la plus haute de la hiérarchie) et les interfaces Entity pour la manipulation des entités considérées comme des **TopiaEntity**. Il y a deux points d'extensions possible pour la génération :

- **DAOImpl** : il est possible de rajouter des méthodes au DAO, ainsi la classe DAOImpl ne sera pas généré et devra être implémenté par le développeur.
- **EntityImpl** : sur le même principe que le DAO, il est possible d'ajouter de nouvelles méthodes à l'entité. Ainsi la classe EntityImpl ne sera pas généré et devra être implémenté par le développeur.

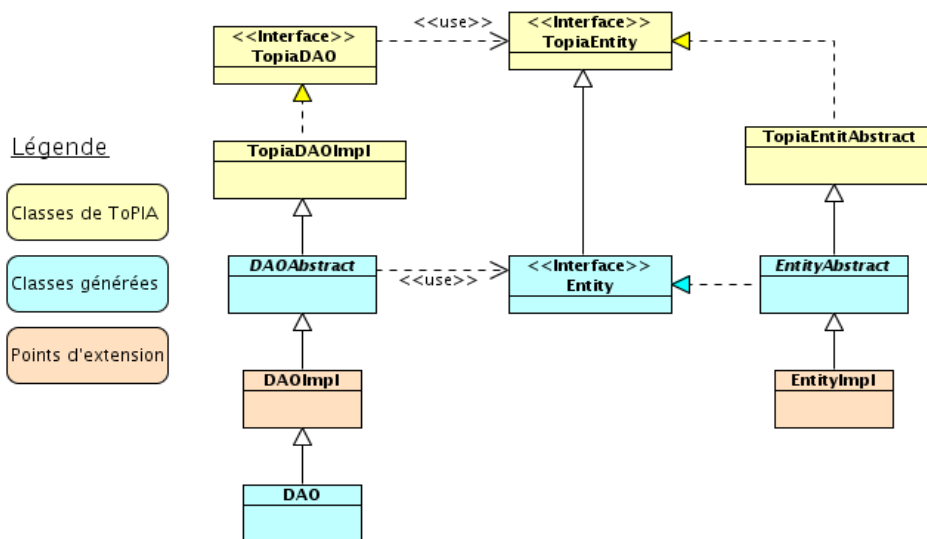


Illustration 7: Hiérarchie des entités et daos générés par ToPIA

Une autre partie de **ToPIA** permet de générer également les services de l'application. Le protocole de communication permettant l'exposition des services peut être configurable via un fichier de propriétés. L'implémentation des services faites dans les différentes applications réalisées utilise la génération des services avec **ToPIA-SOA** mais l'utilisation des services reste simple sans prise en compte du protocole de communication. Les services sont directement accédés depuis le module « ui » de l'application ou via **ChoReg**.

Des **DTO** (Data Transfert Object) peuvent être également générés via **ToPIA** pour permettre la simplification de la manipulation des objets au niveau des services. Ainsi les services manipulent des entités et daos mais communiquent avec le reste du monde via les **DTO** qui seront également générés par **ToPIA**.

Nous allons maintenant voir un exemple avec l'utilisation de **ToPIA** dans l'application Bonzoms.

## b) Exemple de Bonzoms-business

Bonzoms permet la gestion des personnes et sociétés, leurs rôles, relations et contacts. Une société comme une personne sont considérés de la même manière dans l'application. Ce sont des « Party » (ou tiers). Ainsi PartyEntity est la classe centrale de l'application. Une PersonEntity pour décrire une personne, au même titre qu'une GroupPersonEntity pour décrire un groupe de personnes (société, association, ...) héritent toutes les deux de PartyEntity qui possède un type défini par PartyType. Chaque PartyEntity possède plusieurs ContactEntity défini par leurs ContactType (Email, telephone, site internet, ...). L'adresse nécessite plus d'attributs qu'une ContactEntity, ainsi une AddressEntity hérite de cette dernière pour définir plus précisément une adresse comprenant la ville (CityEntity) qui comprend le pays (CountryEntity). Chaque PartyEntity est également composée de plusieurs PartyRoleEntity défini par leurs PartyRoleType (Developpeur, Client, Chef de projet, Directeur, Employeur, ...). Les relations entre les party se fait via leurs rôles. Ainsi chaque RelationshipEntity est lié à deux PartyRoleEntity. Le sens de la relation se fait par rapport à la relation de parenté qu'il apporte. Par exemple, la relation d'Emploi entre une société et une personne est défini comme tel : Un Employé (fromRole) vers son Employeur (toRole) donne la relation Emploi.

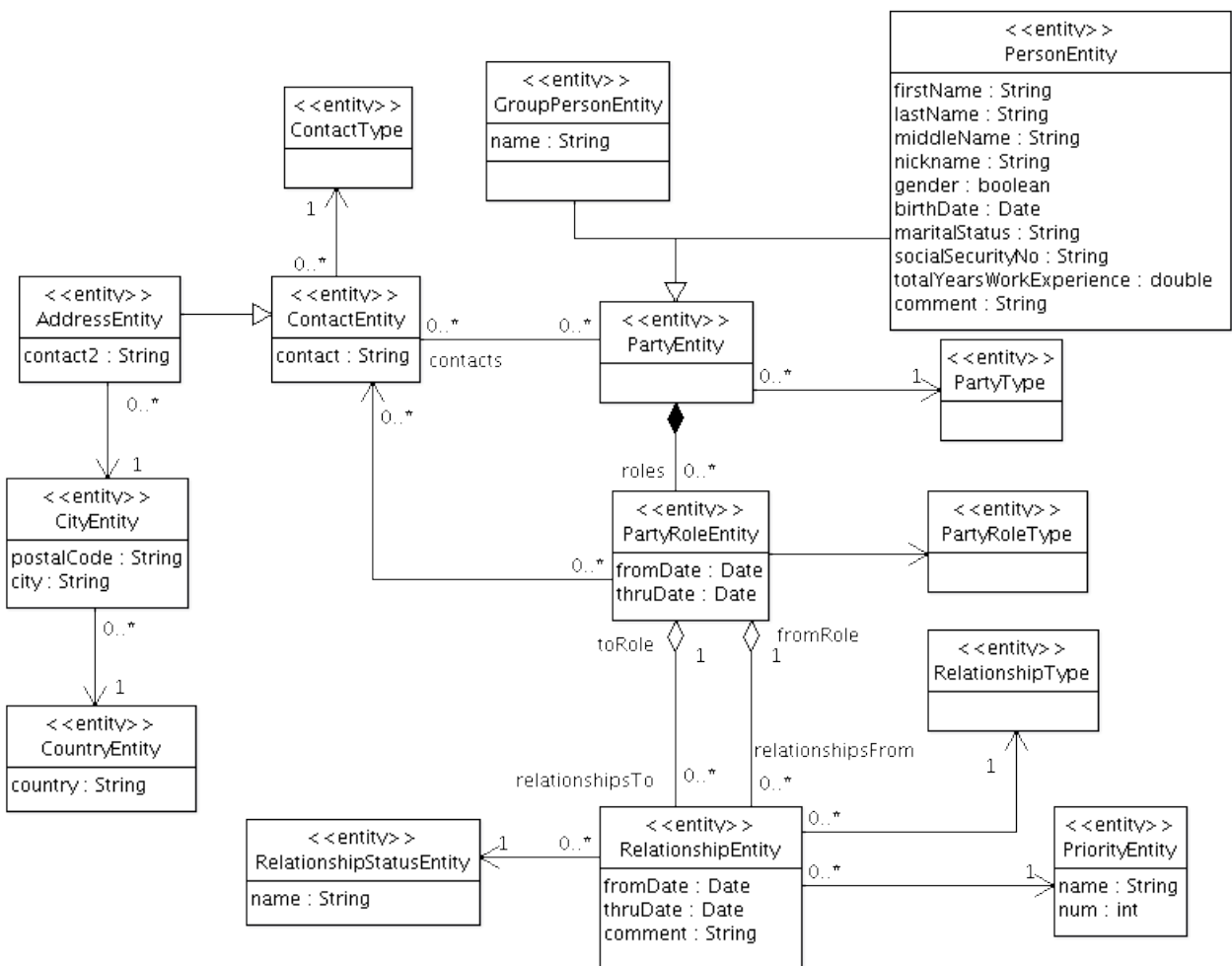


Illustration 8: Modèle du domaine de Bonzoms

Maintenant que nous avons explicité les particularités du modèle de l'application, nous allons décrire la génération des différents éléments permettant de créer la base du métier de l'application : les Entités, DAOs, DTOs et interfaces des Services.

Trois diagrammes sont nécessaires pour cette génération :

- **Entités** : Ce diagramme est équivalent à celui précédent (Illustration 8) avec l'ajout des méthodes sur les entités et les DAOs (Ce qui implique l'implémentation manuel des classes DAOImpl et EntityImpl sur ces entités).

*(Voir diagramme sur l'Annexe p. 3)*

- **DTO** : Les DTO sont générés comme de simples objets Java. Il n'y a aucune dépendance avec ToPIA sur les classes générés (contrairement aux entités). Néanmoins pour simplifier la génération des DTO, il est possible de les faire dépendre d'une ou plusieurs entités. En utilisant la dépendance UML, ToPIA saura que le DTO a un lien fort avec l'entité qu'il dépend. Ainsi le DTO généré aura l'ensemble des attributs de types primitifs de l'entité (String, boolean, int, Date, ...). Cette génération permettra également de simplifier les conversions entre un DTO et une Entité sur les attributs communs aux deux objets. *(Voir diagramme sur l'Annexe p. 4)*
- **Services** : Deux fichiers sont générés pour chaque service modélisé. Chaque service aura une interface, une classe abstraite et une classe d'implémentation qui devra être créée par le développeur. Les services ont uniquement des type DTO dans leurs signatures de méthodes. Chaque méthode de service renvoie également une BonzomsException pour chaque erreur survenue pendant le traitement des données. *(Voir diagramme sur l'Annexe p. 5)*

La génération de Bonzoms avec ToPIA permet de simplifier de nombreux éléments. De plus, il est plus facile de manipuler des diagrammes UML pour ajouter de nouveaux attributs, méthodes ou liaisons entre les objets. La seule partie qui reste à implémenter manuellement est l'implémentation des services de l'application et si besoin, des ajouts sur les entités et DAOs.

### c) Contributions

Durant le stage, plusieurs ajouts ont été effectués sur les projets **EUGene** et **ToPIA** pour les besoins propres au projet Chorem ou pour les besoins d'autres projets de Code Lutin :

A été ajouté dans EUGene : la gestion des énumérations Java depuis l'UML + quelques ajouts sur certains attributs UML (notamment les dépendances UML) non pris en compte lors de la génération des objets Java (Object Model).

A été ajouté dans ToPIA : la génération des DTO avec gestion des dépendances UML + une simplification de la génération de la hiérarchie des DAO, avec prise en compte de l'extension du modèle via le stéréotype « dao » sur une méthode d'une entité (exemple sur le diagramme de l'annexe p. 3). Ces contributions ont permis la génération des DTO et l'utilisation du point d'extension sur les DAOs pour ajouter de nouvelles méthodes.

## d) Export des données dans ChoReg

Pour les besoins d'autres applications Chorem, Bonzoms exporte ses données pour que ChoReg puisse les exploiter. Ces données suivent le format xPIL de la norme OASIS CIQ. Plusieurs services de Bonzoms utilisent cette norme pour être exposés et utilisables via ChoReg. Ainsi le ServicePerson de Bonzoms généré par ToPIA servira également pour l'export d'une personne au format xPIL via ChoReg. Pour le moment le seul format est le format xPIL, mais il sera possible à l'avenir d'utiliser d'autres formats d'échanges. Ainsi pour différencier les formats, chacun d'eux aura sa propre implémentation du service, en plus de l'implémentation de base du service utiliser pour l'« ui ». Pour le ServicePerson, deux implémentations :

- **ServicePersonDTO** qui implémente l'interface générée par ToPIA pour décrire le Service et sera utilisé dans le module ui de Bonzoms.
- **ServicePersonXPIL** qui implémente l'interface ChoremService pour pouvoir être utilisé par ChoReg.

Afin d'optimiser l'implémentation, la partie commune aux deux implémentations est placée dans une classe abstraite utilisant les génériques de Java : ServicePersonGeneric<T>. L'opération « convert » de cette classe est abstraite et devra être implémentée par les deux classes ServicePersonDTO et ServicePersonXPIL pour convertir une donnée de type Entité (généré par ToPIA pour la manipulation de la base de données) en une donnée du type de format de sortie pour l'implémentation du service (DTO ou XPIL). Cette méthode d'implémentation devra être utilisé pour chaque Service exposant ses données auprès de ChoReg.

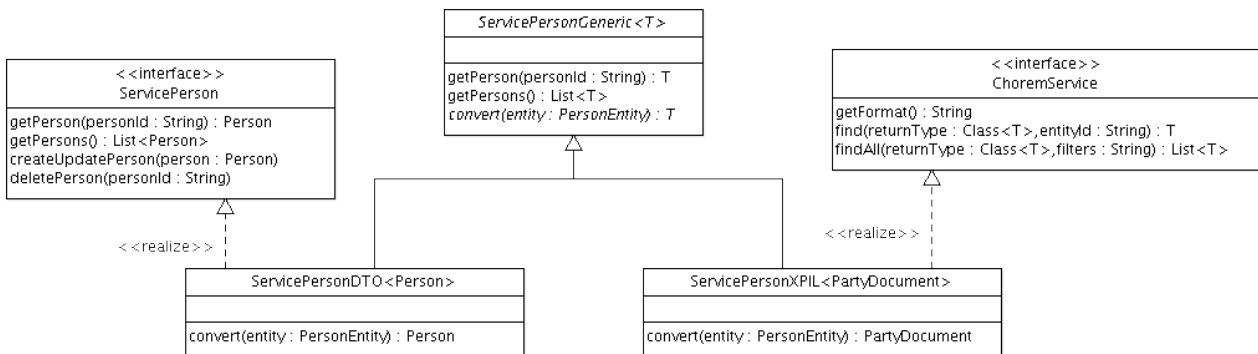


Illustration 9: Diagramme d'implémentation du ServicePerson de Bonzoms

Nous allons maintenant vous présenter le framework Tapestry et son utilisation dans l'interface utilisateur web de Bonzoms.

## 3 - Framework Tapestry

### a) Présentation

Tapestry est un framework java permettant de manière simple de créer une application web avec une architecture à base de composants sans usage de fichiers XML pour la configuration des pages. Tapestry est un projet Apache à l'instar du serveur HTTP (projet Httpd) ou du serveur d'application Tomcat. Le principe de Tapestry réside dans l'utilisation de conventions d'écritures plutôt que de la configuration XML comme Struts (un autre projet Apache) (« Convention over Configuration »).

Chaque page est composée de deux fichiers :

- Le fichier de template (extension .tml) représentant la vue de la page au format XHTML avec des espaces de noms propres à Tapestry.
- Le fichier Java représentant le contrôleur de la page, de ses éléments dynamiques et de ses évènements (clic sur un bouton, chargement de la page, gestion d'un formulaire).

Ces deux fichiers communiquent par convention sur les noms des attributs et méthodes. Chaque fichier de template utilise les composants définies par Tapestry comme élément html de la page. Ainsi les éléments HTML connus sont redéfinis (notamment les éléments de formulaires « input ») et permettent une meilleure prise en charge au sein du fichier java. L'identifiant d'un composant sert de clé pour l'identifier dans le fichier java.

Par exemple pour un lien (composant `actionLink` en Tapestry) :

```
<a t:type="actionlink" t:id="formulairePersonneLink"> Afficher formulaire </a>
```

sera identifié en java comme ceci :

```
private PageLink formulairePersonneLink;
```

De plus le composant **actionlink** renvoie un évènement « action » au moment du click de l'utilisateur. Ainsi la classe java correspondant à la page où se situe le lien pourra récupérer cet évènement via la méthode « *onActionFromFormulairePersonneLink()* ». Il n'est pas nécessaire d'expliquer la liaison tellement le nom de la méthode permet clairement d'identifier l'évènement et le lien auquel elle se rapporte.

## Différences entre les classes java utilisés dans Tapestry

- Page : La classe correspondant à la page (une classe par page) → *package pages*
- Composant : Un composant est une petite partie de la page, généralement un block pouvant être réutilisé. Chaque composant a généralement un fichier template associé pour définir la vue de celui ci. → *package components*
- Mixin : Un mixin permet de rajouter un comportement à un composant (de nouveaux évènements / attributs / actions JavaScript, ...). Le principe du mixin est qu'il est potentiellement applicable à n'importe quel composant. → *package mixins*
- Service : Un service est une classe distincte permettant un traitement sur des données. Il ne concerne pas l'affichage mais seulement la manipulation des données (persistance, gestion des langues, traitement externe (web service), ...) → *package services*

Tapestry se différencie des autres framework web java par son approche modulaire et dynamique nécessitant aucune configuration XML. Ainsi chaque partie de l'application sera conçu indépendamment sous forme de composant. On peut ainsi considéré chaque page comme un composant pouvant communiquer avec une autre page ou un composant fils ou père. Le cœur de tapestry est basé sur le principe d'inversion de contrôle (ou injection de dépendance) qui en régit l'architecture principale du framework. Ainsi les objets manipulés par Tapestry sont injectés dans les différentes pages, composants et services qui composent l'application web. Les composants et pages sont par défaut injectable via différentes annotations Java :

- @Inject : injection d'un élément (principalement les services)
- @InjectPage : injection d'une autre page
- @InjectComponent : injection d'un composant de la page (inclut dans le template)
- @InjectContainer : injection du conteneur du composant (généralement une page ou un autre composant)

Nous allons maintenant décrire le module « ui » de l'application Bonzoms qui utilise le framework Tapestry.

## b) Exemple de Bonzoms-ui (IHM)



L'application Bonzoms est composé de trois parties distinctes :

- Gestion des personnes : page de vue d'une personne, page du formulaire pour l'ajout/modification d'une personne et page de liste de toutes les personnes.
- Gestion des sociétés : page de vue d'une société et page de liste de toutes les sociétés
- Gestion des rôles et relations : une seule page permettant la sélection d'un tiers (personne ou société) et de gérer précisément ses rôles et relations.

Nous allons décrire les deux principales pages, celle de la liste et celle de la vue (détails). Les pages sont très proches que ce soit pour une personne ou une société.

### Page Liste

Les pages de liste (une pour les sociétés et une pour les personnes) sont basé sur le composant Tapestry *Grid* qui permet d'afficher un tableau représentant l'ensemble des éléments d'un type donné. Ici il s'agit de la liste des personnes de type *Person*. Chaque ligne du tableau représente les informations principales d'une personne (prénom, nom, sexe, date de naissance, expérience). Pour chaque ligne, l'utilisateur peut :


- Modifier la personne : une fenêtre apparaît comprenant le formulaire d'une personne avec les champs remplis. 
- Supprimer la personne : un message de confirmation demande à l'utilisateur de confirmer sa demande de suppression. 
- Afficher les détails de la personne en cliquant sur le nom ou prénom de la personne: la page des détails de la personne apparaît.



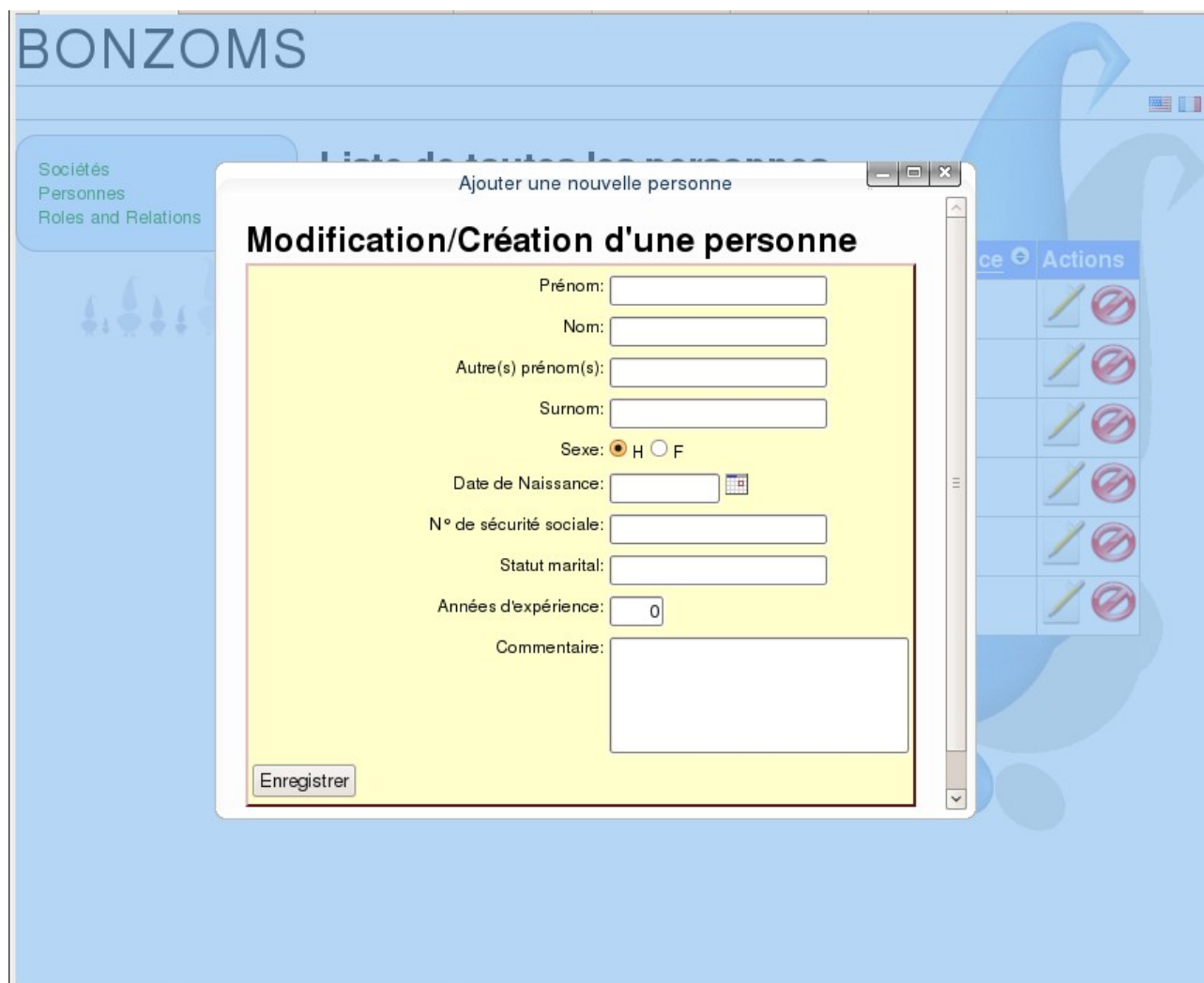
The screenshot shows the BONZOMS application interface. At the top left, the title 'BONZOMS' is displayed. Below it, there is a navigation menu with three items: 'Sociétés', 'Personnes', and 'Roles and Relations'. The 'Personnes' item is selected. The main content area is titled 'Liste de toutes les personnes' and features a table with the following data:

Prénom	Nom	Sexe	Date de Naissance	Expérience	Actions
<a href="#">Eric</a>	<a href="#">Chatellier</a>	H		3.0	 
<a href="#">Arnaud</a>	<a href="#">Thimel</a>	H		5.0	 
<a href="#">Nolwenn</a>	<a href="#">Rannou</a>	H	2 févr. 1985	0.0	 
<a href="#">Kevin</a>	<a href="#">Morin</a>	H	30 juil. 1984	0.0	 
<a href="#">Florian</a>	<a href="#">Desbois</a>	H	3 mars 1985	0.0	 
<a href="#">Benjamin</a>	<a href="#">Poussin</a>	H		10.0	 

Illustration 10: IHM Bonzoms - Liste des personnes

L'utilisateur peut également ajouter une nouvelle utilisateur en utilisant l'icône 

Comme pour la modification, un formulaire apparaît pour remplir les informations de la personne (Une seule page pour ajout/modification d'une personne) :



*Illustration 11: IHM Bonzoms - Ajout nouvelle personne*

La liste des sociétés fonctionne sur le même principe de tableau (Grid) avec icônes d'ajout/modification/suppression. Cependant il n'y a pas de fenêtre pour le formulaire, seul le nom de la société est considéré et le champ texte se situe directement à côté de l'icône d'ajout.

## Page Details (View)

La page de détails est différente pour une personne ou une société. Cependant le principe reste identique. La page est composé d'une partie représentant les attributs principaux du tiers (personne ou société) et de différents composants de gestion :

- Pour une société : Contacts, Services, Employés, Rôles et Relations
- Pour une personne : Contacts, Rôles et Relations

Les parties communes aux deux pages sont donc devenues des composants Tapestry :

- ContactComponent : gestion des contacts pour un tiers (party) que ce soit une personne ou une société.
- RoleComponent : gestion des rôles pour un tiers.
- RelationComponent : gestion des relations pour un tiers.

Pour permettre aux composants de communiquer facilement avec leur page conteneur, une classe abstraite et une interface ont été créés. Les deux classes *OrganizationView* et *PersonView* représentant les pages de détails respectivement d'une société et d'une personne, héritent toutes deux d'une même classe abstraite « *AbstractPartyView* » qui implémente l'interface « *PartyView* ». Les comportements communs des deux pages sont regroupés dans la classe abstraite. L'interface permet d'identifier le type de page auprès des composants qui en ont besoin. Ainsi les méthodes *getParty()*, *getPartys()* et *getPartyId()* seront utilisables dans tous les composants inclus dans les pages *OrganizationView* et *PersonView*. Comme le montre le diagramme, le composant *RoleComponent* utilise l'injection *@InjectContainer* pour connaître la page dans laquelle il se situe qui est obligatoirement une *PartyView*.

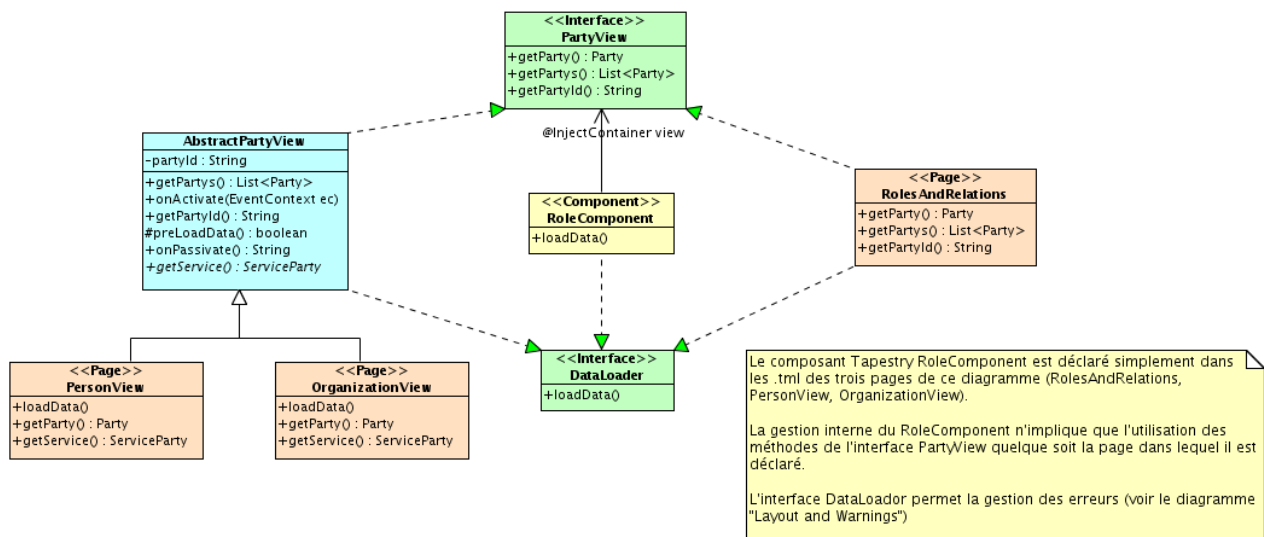


Illustration 12: Diagramme d'implémentation de PartyView

A noter l'utilisation du DataLoader pour la gestion des messages d'erreurs au chargement des pages.

L'intérêt de cette architecture est d'utiliser des composants indépendamment de leurs pages à condition qu'elles respectent l'interface utiliser par le composant. Ce qui est particulièrement appréciable dans Bonzoms vu que beaucoup d'éléments sont communs suivant les pages comme la gestion des rôles et la gestion des relations.

Voici la page des détails pour une société. Les composants peuvent être ouverts ou fermés, ici le composant ContactComponent est ouvert pour laisser apparaître la liste des contacts de la société Code Lutin. Les composants de gestion des services et des employés sont cachés. Dans cette version les composants de gestion des rôles et relations n'ont pas été ajoutés, cependant il suffit d'une ligne dans le fichier de template pour le permettre (`<div t:type="rolecomponent" />` pour le composant de gestion des rôles).

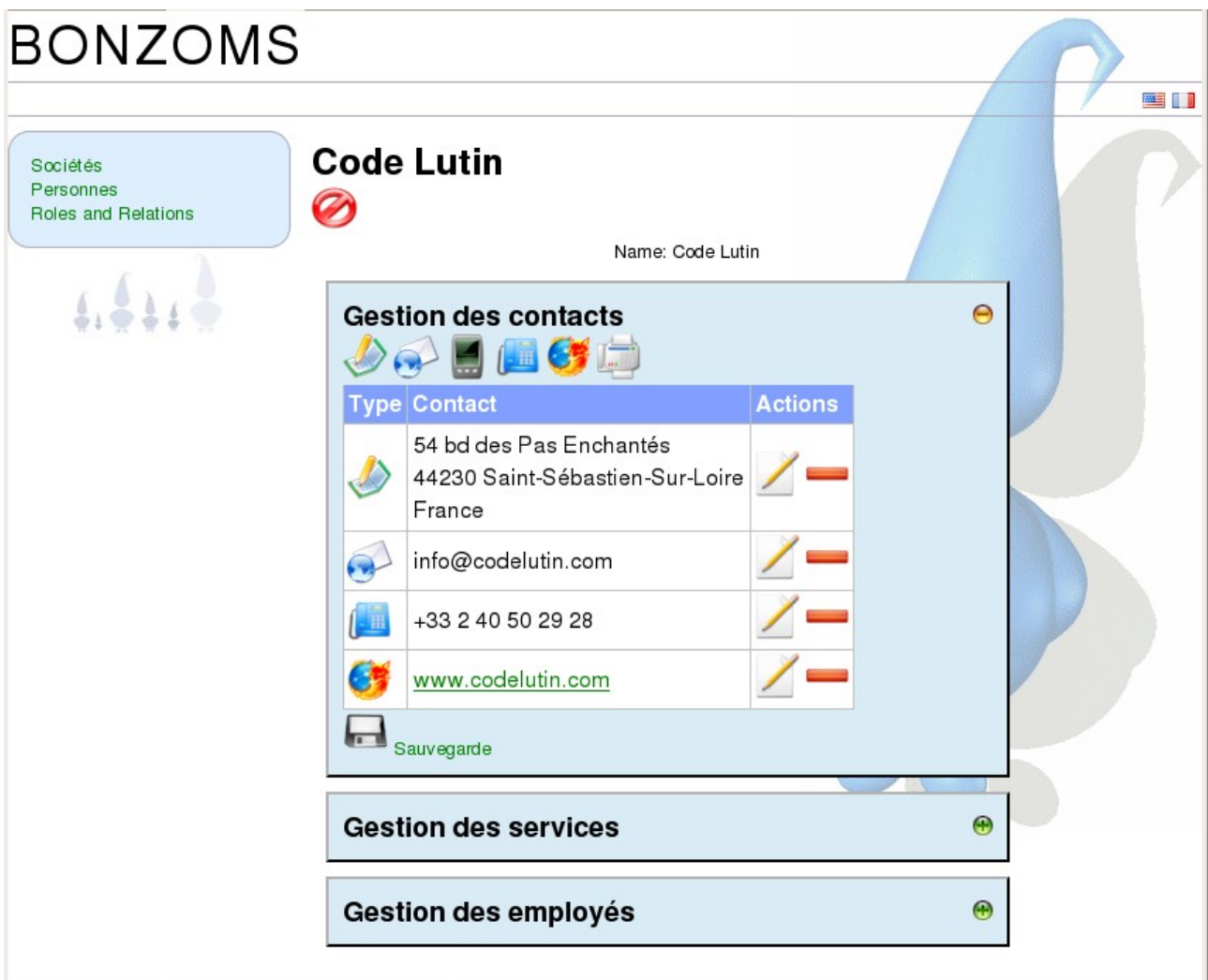


Illustration 13: IHM Bonzoms - Détails d'une société avec contacts ouvert

Le composant de gestion des contacts est dynamique : toutes les actions utilisateurs sont faites sans sauvegarde en base de données. L'icône permet la sauvegarde réelle des changements.

L'ajout d'une ligne de contact s'effectue via la liste des icônes au dessus du tableau. Chaque icône représente un type de contact (adresse, email, téléphone, internet,

fax). Tous ces types n'ont qu'un seul attribut excepté le type adresse, plus complexe, qui contient des champs pour la ville, le code postal et le pays. Ces derniers n'existent pas au préalable en base de données, ce sont de simples champs textes. Pour aider l'utilisateur, une autocomplétion est mise en place sur le champs de Ville (Si la ville existe déjà, aucun intérêt à la recréer). Plusieurs propositions apparaissent suivant les villes : plusieurs code postaux, etc... Une fois sélectionné dans la liste, les champs ville, code postal et pays sont automatiquement chargés dans le formulaire.

The screenshot shows the BONZOMS web application interface. At the top left, the logo 'BONZOMS' is displayed. Below it, a navigation menu includes 'Sociétés', 'Personnes', and 'Roles and Relations'. The main header area shows 'Code Lutin' with a red prohibition sign and the text 'Name: Code Lutin'. A large blue hat graphic is visible in the background. A modal window titled 'Gestion des contacts' is open, displaying a table of contact information for 'Code Lutin'.

Type	Contact	Actions
	Adresse : <input type="text" value="54 bd des Pas Enchantés"/> Complément d'adresse : <input type="text"/> Ville : <input type="text" value="Saint-Sébast"/> <b>Saint-Sébastien-Sur-Loire - 44230 - France</b>	
	info@codelutin.com	
	+33 2 40 50 29 28	
	<a href="http://www.codelutin.com">www.codelutin.com</a>	

At the bottom of the modal window, there is a 'Sauvegarde' button with a floppy disk icon.

Illustration 14: IHM Bonzoms - Détails d'une société avec édition des contacts

## IV – Gestion de projet

---

### 1 – Organisation du projet

Les besoins du projets et les objectifs ont de nombreuses fois évolués au cours du processus de développement. L'architecture n'a pas été rapidement trouvé. Le début du stage a surtout permis de faire des recherches et de se former sur les différentes technologies utilisées pour l'implémentation des applications ainsi que l'architecture (comme par exemple les essais sur la technologie OSGi). A la fin de chaque semaine devait être remis un compte rendu permettant de détaillé le travail effectué sur la semaine, les problèmes rencontrés et le travail à fournir pour la semaine suivante. Ces comptes rendus ont permis d'avancer étape par étape sur les besoins du projet, notamment ceux des différentes applications.

### 2 - Problèmes rencontrés

Le sujet étant vaste, quelques problèmes d'organisation sont survenus pendant le stage. Le modèle de données de la première application réalisée, Bonzoms, fut complexe à mettre en oeuvre. En effet, le modèle étant très générique, de nombreux utilitaires et méthodes spécifiques pour simplifier son utilisation ont dûs être créés. De plus, l'apprentissage du framework Tapestry pour réaliser l'interface a pris plus de temps que prévu. Mais ce premier projet a permis de poser des bases solides pour la conception des autres applications Chorem non existantes.

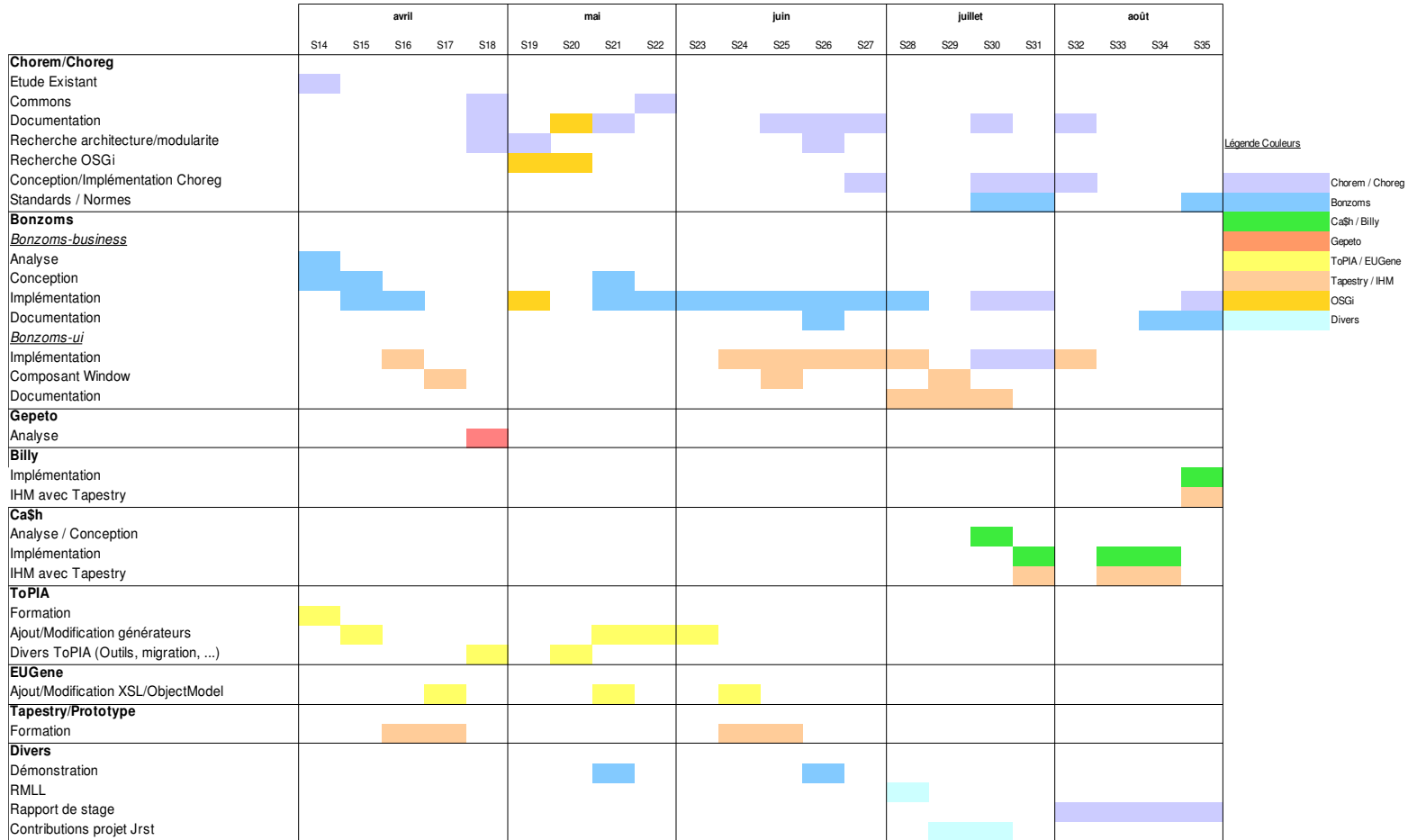
En parallèle, la recherche d'une architecture pour la communication entre les projets a été difficile. Le monde Java est très vaste et de nombreuses technologies existes permettant de mettre en place une architecture comme celle-ci. Après quelques semaines de recherche et de nombreux essais sur le framework OSGi, une solution plus simple a été mise en place sans prise en compte de la distribution des applications.

Une fois l'architecture trouvée et Bonzoms terminé, l'implémentation et les tests d'architecture furent très rapides (1 semaine + 3 jours de documentation), de même que l'implémentation de l'application Ca\$h (4 jours). Néanmoins, c'est en fin de stage qu'il s'est avéré que la communication ne se fait pas entre Bonzoms et Ca\$h mais en passant par la gestion des factures, donc Billy, qui a été rapidement mis en place.

Le temps passé au début du stage sur la recherche et la formation (ToPIA, Tapestry), n'a pas permis de terminer la communication entre les trois applications Bonzoms, Ca\$h et Billy via ChoReg. Cependant, ChoReg est opérationnel, Bonzoms l'utilise pour exporter ses données, Ca\$h est complètement opérationnel pour une utilisation seul (stand-alone) et Billy peut utiliser les données proposées par Bonzoms. De plus, de nombreuses documentations ont été rédigés pour permettre l'utilisation de ChoReg ou faciliter la reprise des projets pour de futurs développeurs / stagiaires (voir bibliographie pour les liens vers les documentations créées).

En parallèle du projet, plusieurs contributions ont été effectués dans les projets de Code Lutin comme EUGene, ToPIA ou JRst, pour des besoins du projet ou des demandes d'autres développeurs. Vous trouverez le planning réel du stage sur la page suivante.

### 3- Planning réel



## Conclusion

---

Les applications Chorem créés sont toutes indépendantes et peuvent, si d'autres applications sont présentes dans le même serveur d'application, exporter ou importer leurs données les unes avec les autres. ChoReg permet ces échanges suivant des normes internationales de format des données. Les contraintes de l'architecture ont été respectés par ChoReg et trois applications l'utilisent avec succès.

Concernant l'architecture interne de chaque application, les deux frameworks ToPIA et Tapestry ont bien été suivis et exploités de manière appropriée et performante. Nous avons donc une architecture solide pour trois applications : Bonzoms, pour la gestion des personnes, Ca\$h pour la gestion de la trésorerie et Billy pour la gestion des factures. Même si les parties métiers et services des trois applications ne sont pas complètes, ces dernières ont toutefois une base solide pour la suite du développement.

Concernant le stage, malgré les quelques problèmes d'organisation rencontrés, l'objectif du projet a été atteint. L'ambiance de travail et l'aide apportée pendant toute la durée du stage par les autres développeurs lutins a permis de concrétiser l'architecture et poser les bases solides des premières applications de gestion d'entreprise de la plateforme Chorem.

D'autres applications doivent être maintenant réalisées pour compléter les fonctionnalités des trois applications créées et des trois applications déjà existantes. Ainsi le projet Gepeto concernant la gestion de projet et le projet Kalendaro concernant la gestion d'un calendrier partagé sont deux applications intéressantes pour la suite du développement et l'utilisation plus en profondeur des bénéfices d'une architecture comme celle de ChoReg.

## Bilan personnel

---

Le stage a été une source très enrichissante de connaissances divers et variés dans le monde Java. Les trois premiers mois m'ont permis d'apprendre de nombreuses techniques de développement et de nombreuses technologies Java (JMS, MOM, JBI, OSGi, Tapestry, ServiceMix, Tomcat, XML, XSL, Hibernate, EUGene, ToPIA, Jrst...). La communauté libre et l'esprit de cette philosophie dans la société Code Lutin ont également été des sources très enrichissantes de savoir vivre et de savoir faire.

Mes études sont désormais terminées avec cette dernière année de Master 2 ALMA qui m'a permis de perfectionner mes connaissances en Génie Logiciel et Architecture Logicielle. Je suis maintenant prêt à rentrer dans le monde du travail et ceci se concrétise immédiatement grâce à la société Code Lutin qui m'emploie après un peu de vacances bien mérité !

## Bibliographie

---

### Modèle de données :

- ERP Ofbiz Neogia : <http://www.neogia.org/Accueil>
- The Data Model Resource Book (A Library of Universal Data Models for All Enterprises) de *Len Silverston* (2001)

### Outils de développement :

- Maven : <http://maven.apache.org/>
- Tutoriel Maven : <http://dcabasson.developpez.com/articles/java/maven/introduction-maven2/>
- Subversion : <http://subversion.tigris.org/>
- NetBeans : <http://www.netbeans.org/>

### Projets Code Lutin :

- Forge ChoreM (juin 2009) : <http://chorem.org>
- Documentation EUGene : <http://nuiton.org/embedded/eugene/eugene/index.html>
- Documentation ToPIA : <http://nuiton.org/embedded/topia/index.html>
- Documentation JRst : <http://nuiton.org/embedded/jrst/jrst-doc/index.html>

### Liens pour interface web :

- Apache Tapestry : <http://tapestry.apache.org/tapestry5.1/>
- Librairie JavaScript Prototype : <http://www.prototypejs.org/>
- Composant Window prototype : <http://prototype-window.xilinus.com/>
- Librairie composants Tapestry : <http://www.chenillekit.org/>

### Recherche architecture :

- OSGi Alliance : <http://www.osgi.org/Main/HomePage>
- Définition OSGi Wikipedia : <http://fr.wikipedia.org/wiki/OSGi>
- Apache Felix : <http://felix.apache.org/site/index.html>
- Apache ServiceMix : <http://servicemix.apache.org/home.html>
- JMS : <http://java.sun.com/products/jms/>

- Définition JBI Wikipedia : [http://fr.wikipedia.org/wiki/Java\\_Business\\_Integration](http://fr.wikipedia.org/wiki/Java_Business_Integration)
- Blog de Kirkk (articles sur OSGi) : <http://techdistrict.kirkk.com/>
- OSGi – The Dynamic Module System for Java : Part 1 by *Aneesh Kumar* (27/01/2009) : <http://aneeshkumarkb.blogspot.com/>

Normes / Standards :

- Norme Oasis CIQ : [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ciq](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ciq)

Projets réalisés :

- Documentation Bonzoms : <http://chorem.org/embedded/bonzoms/index.html>
- Documentation ChoReg : <http://chorem.org/embedded/choreg/choreg/index.html>
- Projet Ca\$h : <http://chorem.org/projects/show/cash>