Système de Virtualisation pour une application de gestion commerciale d'entreprise

Travail d'Etude et de Recherche
Master 1 STIC Informatique

ETUDIANTS

FIDAN AYHAN
SAHLOUL SAHBI
BOSQUET SYLVAIN
COUNDOUL IBRAHIMA

CHEF DE PROJET

BOSQUET SYLVAIN

ENCADRANT

PIERRE GUILLOT

Année Universitaire 2006–2007

Table des matières

Remerciements

Nous tenons à remercier M. Pierre Guillot, notre encadrant, pour nous avoir permis de réaliser ce TER, pour son accueil dans l'entreprise, pour son dévouement et sa compétence.

Nous remercions également M. Fabrice Huet pour avoir accepté un sujet externe à l'université, qui nous a permis d'accéder à la vie réelle d'une entreprise d'informatique.

Merci à M. Philippe Collet pour la pertinence de ses conseils sur la rédaction du cahier des charges.

1. Introduction

Ce rapport est le document final du travail d'étude et de recherche du Master 1 Informatique à l'Université de Nice Sophia-Antipolis. Il concerne la conception d'un système de virtualisation[1] d'une base de données existante pour une application de gestion commerciale d'entreprise[2]. Ce système est réalisé en Java.

Ce sujet est à l'initiative d'un étudiant de l'université. Il répond à un besoin applicatif d'une entreprise. Le sujet est donc réalisé en collaboration entre l'entreprise SICA (Solutions Informatiques Cote d'Azur) et la faculté de Sciences de Nice Sophia Antipolis.

L'entreprise a mis à disposition des étudiants un espace de travail, des outils et du matériel. Le lieu de travail est donc défini au siège de l'entreprise (3 rue Poincaré).

Le but de ce projet est de concevoir et de réaliser un système de virtualisation / d'abstraction complexe d'une base de donnée, afin de pouvoir y travailler à distance. Cette couche de virtualisation est utilisée par un logiciel client[3] permettant l'édition de documents de gestion commerciale (devis, factures, bon de commande…).

1.1. Contexte de développement

Avant de commencer à travailler, nous avions posé certaines hypothèses. Une seule ne fut pas valide : Le temps accordé au déroulement du TER n'est pas suffisant. Deux facteurs en sont la raison : Un mois de travail à temps plein est insuffisant ; Tous les étudiants travaillant à ce projet sont salariés, le temps qu'ils peuvent consacrer au projet est plus faible que les autres.

Les projets de l'entreprise relatifs à la base de données Oracle sont influents sur le déroulement du TER. Une coordination avec le personnel de l'entreprise prenant par à ces projets est indispensable. De même, certaines conditions posées dans le cahier des charges tendent à rendre le déroulement du TER difficile : Il est impossible de modifier la structure de la base. Il est indispensable de respecter les cohérences de données dans la base. De même, il nous a été imposé l'utilisation d'outils libres de droits et gratuits.

1.2. Limites du développement

Il est important de définir précisément les limites de développement pour la compréhension de ce document.
La gestion de la base ne fait pas partie du TER. Les projets relatif à cette base interviennent dans le déroulement du TER, mais n'en font pas partie. De même, la compréhension du fonctionnement de cette base ne fait pas non plus partie du TER. Autrement dit, la configuration du mapping est effectué par du personnel de l'entreprise.
Mr Sylvain Bosquet, salarié de l'entreprise, est intervenu massivement dans les projets relatifs à la base Oracle, dans le cadre de son travail, pas dans le cadre du TER.

[1] Système logiciel d'abstraction du fonctionnement d'une base de données.
[2] Gestion de pièces commerciales (Devis, factures…)
[3] Logiciel se connectant à un serveur d'application

Avant de détailler le déroulement de notre TER, nous rappellerons les points essentiels de notre cahier des charges afin que la mise en situation soit complète. Ensuite, nous verrons dans un premier temps comment nous avons abordés notre travail, puis quels sont les problèmes rencontrés et les solutions que nous avons été amenés à choisir. Et enfin nous préciserons où nous en sommes aujourd'hui de la réalisation du projet. Pour finir, nous conclurons sur ce projet.

2. Cahier des charges

Le cahier des charges rendu lors de la pré-soutenance, est disponible à l'adresse suivante : http://ter.atm-net.org/uploads/Main/cahier.pdf - Ce qui suit, est un résumé de ce cahier des charges.

2.1. Présentation générale du problème

Concevoir un système de virtualisation pour une application de gestion commerciale d'entreprise sous base Oracle en Java.

Les finalités sont les suivantes :

- Virtualisation par l'intermédiaire de serveur front office[4] (linux, Java).
  - Mapping[5] configurable objet/relationnel (introspection[6] inférante de la base)
  - Initialisation du serveur front office avec l'arbre de configuration
- Logiciel client java
  - Intranet/Internet
  - Gestion de pièces commerciales
  - Gestion des fichiers de la base

La première finalité est une étape d'auto-configuration du système. La seconde est la finalité de la suite logicielle.

2.2. Contexte du problème

La base de données Oracle est existante, elle est en utilisation permanente dans l'entreprise. Les données contenues dans cette base sont qualifiées de critiques.

Situation du projet par rapport aux autres projets de l'entreprise :

La base de données a été migrée d'Oracle 8 vers Oracle 10g XE le 06/03/2007. Cette dernière est en cours d'utilisation par une solution logicielle d'éditeur propriétaire dont la gestion de la base est jugée globalement insatisfaisante. D'où la nécessité de revoir la gestion globale de l'administration de la base.

Etudes déjà effectuées :

Evaluation de la solution logicielle en vu d'une utilisation aménagée.
Exploration de la base de données et extraction d'informations de structures, ainsi qu'un rapport des non conformités SQL[7].

[4] Gestion des accès, guichet
[5] Procédé de virtualisation par association des données avec des tables de correspondances
[6] Analyse de soi
[7] Langage de requêtes pour les bases de données

2.3. Nature des prestations demandées

Les prestations ont été effectuées selon deux phases :

Une première phase d'étude :

Etude des technologies existantes

Introspection d'une base de données Oracle

Mapping Objet/relationnel configurable

Etude du schéma pour la structure objet du Client/Serveur Java

Une deuxième phase de développement :

Utilisation des technologies sélectionnées pendant la phase d'étude

Utilisation d'outils d'introspection comme Hibernate[8] Synchroniser[9] et Hibernate Tools[10]

Mapping et codage de l'arbre de configuration

Codage du modèle Client/Serveur

Gestion de pièces commerciales.

Caractère confidentiel :

L'encadrant a mis en avant l'exigence de confidentialité du projet.

2.4. Fonctionnalités

Expression du besoin par l'entreprise:

Mettre à disposition de l'utilisateur des fonctions de haut niveau pour la maintenance des fichiers de la base de production.

Mettre à disposition distante des informations de la base par un client léger et portable[11] (exemple : consultation d'un devis client de manière sécurisé par internet)

Créer un système de virtualisation et de gestion évolué de l'arbre de correspondance pour l'administrateur.

Respecter l'intégrité de la base de production actuelle et maintenir l'isolation galvanique[12] de sécurité de la base.

[8] Framework open source gérant la persistance des objets en base de données relationnelle
[9] Outils Hibernate
[10] Outils Hibernate
[11] Non dépendant du système d'exploitation
[12] Isolation totale

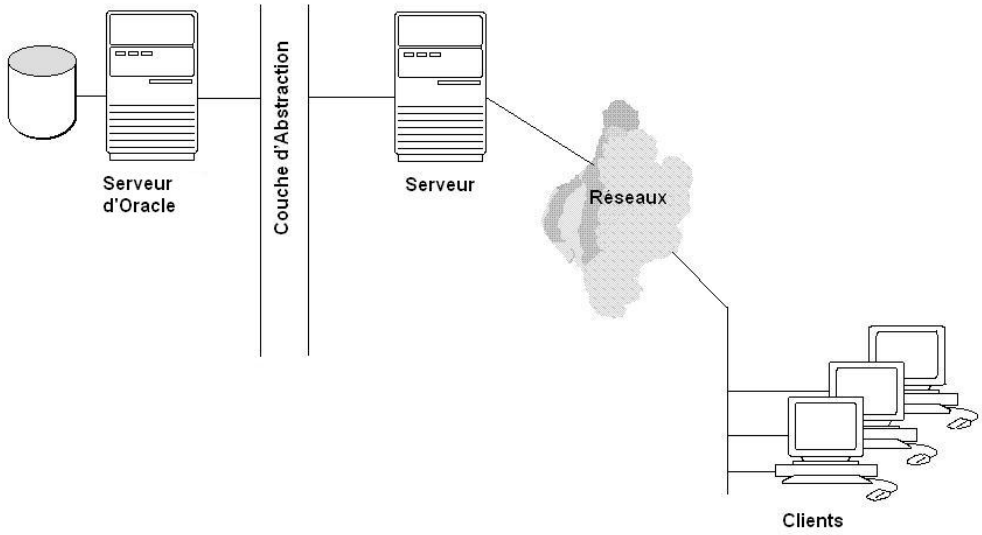

Figure 1. Schéma général

Ci-dessus, figure 1, le schéma général extrait de l'expression des fonctionnalités du système.

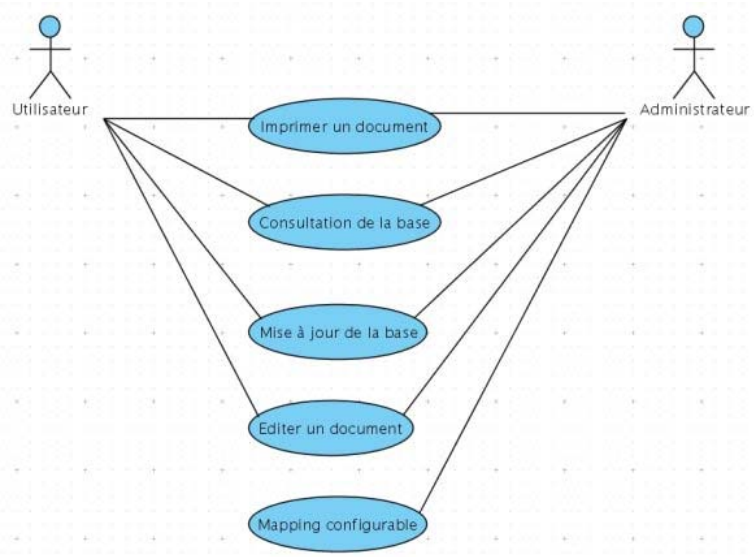

Figure 2. Diagramme général des cas d'utilisations

Ci-dessus, figure 2, le diagramme général des cas d'utilisations extrait de l'expression des fonctionnalités du système.

3. Organisation du projet

A cause du temps très limité annoncé dans le TER, nous avons adopté une démarche incrémentale afin de valider étape par étape le fonctionnement de base du système. Le but est de toujours être en mesure de présenter un système, aussi minimal soit-il, qui fonctionne.
A chaque boucle incrémentale, des tests ont été réalisé, afin de confirmer ou d'infirmer la cohérence du projet avec le sujet. Mais également pour la correction d'éventuels bugs.

Trois parties seront menées en parallèle incrémentalement :
- La virtualisation de la base de données
- Le modèle Client/serveur
- Les outils de généricité[13]

Avant de détailler le fonctionnement et les techniques des trois parties, il est important de définir l'organisation de notre TER.

3.1. Organisation logistique

Comme annoncé en introduction, l'entreprise qui nous a accueillis, à mis à notre disposition une salle de travail et plusieurs machines.
Il parait utile de préciser que M. Sylvain Bosquet, en sa qualité de salarié de l'entreprise, s'est chargé de l'organisation logistique du TER. De par cette double responsabilité, le travail effectif des étudiants a été légèrement retardé. En effet, les deux premiers jours du travail à temps plein furent rythmés par l'installation et la configuration logicielle des machines de travail et du réseau. De même, l'installation tardive des serveurs de test Oracle et java a contribué à pénaliser le travail effectif. Toute l'équipe a participé à ces installations. Cependant, tout est rentré rapidement dans l'ordre, et le travail a pu être réalisé dans de bonnes conditions.

3.1.1. Outils de travail

L'équipe de développement a utilisé un wiki[14] pour mesurer l'avancement du projet. Ce wiki est donc considéré comme bloc note, pour noter les problèmes, les solutions, les rapports et l'évolution du travail.
Elle a également utilisée un calendrier partagé, afin de communiquer, de planifier les tâches avec l'encadrant. L'outil utilisé est horde[15].
De même, l'équipe a utilisé un serveur CVS[16] pour la gestion des sources.
La partie développement a été réalisée en Java à l'aide d'ECLIPSE[17], un IDE[18] libre. Certains étudiants, travaillant sur la partie virtualisation ont été amenés à utiliser une version modifiée d'éclipse, intégrant des outils Hibernate : JBOSS IDE ECLIPSE[19].

[13] Indépendance vis-à-vis du type d'objet manipulé
[14] Système de gestion de contenu de site Web
[15] Logiciel open source pour la gestion des e-mails et des calendriers
[16] Logiciel libre de gestion de versions (Concurrent Versions System)
[17] IDE écrit en java permettant de créer des projets de développement dans n'importe quel langage
[18] Environnement de développement intégré
[19] Version modifiée d'Eclipse optimisée pour Hibernate

Les machines utilisées par les étudiants comme poste de travail pouvaient fonctionner alternativement sous Windows 2000 ou sous Linux Ubuntu[20]. La portabilité du logiciel développé a donc pu être testée.

3.2. Organisation humaine

L'organisation humaine générale avait été prévue dans le cahier des charges de la manière suivante :

- **Virtualisation** : Sahbi Sahloul et Sylvain Bosquet
- **Client/Serveur** : Ayhan Fidan et Ibrahima Coundoul

Cependant, le partage des tâches a été redéfini et modifié selon les besoins. En effet, nous avions sous estimé le code métier[21] du serveur Java. Ce dernier doit être capable de manipuler des objets de manière générique en vue de les transmettre aux clients. Nous avons donc créé une troisième grande partie, au même titre que la virtualisation et le modèle Client/Serveur, à savoir, la partie « Outils de Généricité ».

Les trois parties ont été réalisées en parallèle. Voici donc le partage effectif des tâches :

3.2.1. La virtualisation

De manière générale, cette partie est réalisée part le binôme Sahbi Sahloul et Sylvain Bosquet. Cette partie, contrairement aux deux autres, ne nécessite que peu de code écrit. Une recherche plus importante est primordiale, associé à un savoir faire acquis lors d'expérimentations. Le binôme a effectué ses recherches séparément et mis en commun les résultats lors de réunions. Sylvain a écrit les classes de jointure pour le code métier du serveur et les fichiers de configuration. Sahbi a réalisé un outil d'assistance à la configuration basé sur du DOM[22] XML[23].

3.2.2. Le modèle Client/Serveur

Cette partie est attribuée au binôme Ayhan Fidan et Ibrahima Coundoul. Elle consiste en l'écriture d'un modèle Client/Serveur en Java, ainsi qu'une interface simple pour le logiciel client. Les deux étudiants ont défini ensemble le modèle et les techniques qu'ils utiliseraient lors de l'implémentation. Ayhan a donc implémenté ce modèle. Ibrahima a implémenté de son coté l'interface cliente. Il est à noter qu'Ayhan, Ibrahima et Sylvain ont effectué ensemble la jointure physique des parties virtualisation et Client/Serveur.

3.2.3. Les Outils de généricité

Ici, le découpage du travail est organisé différemment des deux autres parties. En effet, trois personnes ont été indispensables à la réalisation des outils de généricité. Ayhan, Ibrahima et Sylvain ont défini et cherché les techniques de généricité nécessaires afin que le serveur soit

[20] Distribution Linux basée sur debian
[21] Savoir faire du serveur d'application
[22] Document Object Model
[23] Extensible Markup Language

capable de manipuler des objets indépendamment du type. Cette partie est de loin la plus difficile de tout le TER. Ayhan et Ibrahima ont implémenté ces outils.

3.2.4. Rédaction du rapport

Il nous parait utile de définir ici le partage des tâches pour la rédaction du rapport car, celui-ci est une partie importante de la note finale.
Tous les étudiants ont participé à l'élaboration du plan général et à la cinématique des parties. Sylvain a rédigé la partie avant du rapport, à savoir, l'introduction, le rappel sur le cahier des charges, et l'organisation. Sylvain et Sahbi ont rédigé la partie sur le développement de la virtualisation. Ayhan et Ibrahima ont rédigé les parties sur le développement du Client/Serveur et sur les outils de généricité. Ayhan, Ibrahima et Sylvain ont écrit ensemble la partie sur l'état d'avancement de notre projet, mais aussi la partie conclusion et synthèse.

3.3. Planning

Un planning avait été proposé dans le cahier des charges. Malgré les efforts réels de toute l'équipe, le planning a dû être modifié. L'équipe a dû également prendre en compte les obligations des étudiants salariés. Comme tous les étudiants de l'équipe sont salariés, de nombreux aménagements du planning ont dû être réalisés.

3.3.1. Planning réalisé

La durée accordée au plein temps pour le TER est d'environ 5 semaines.

La virtualisation fut réalisée en un peu plus de 2 semaines, l'utilitaire de configuration en quelques jours, après les 2 semaines.
Le Client/serveur fut réalisé en 1 semaine, le savoir faire avait été acquis en Master 1 durant les travaux dirigés.
Par contre, les outils de généricité ont demandés 3 semaines, et trois membres du groupe. C'est dont la partie la plus importante du sujet.
La rédaction du rapport a été réalisée pendant la dernière semaine.

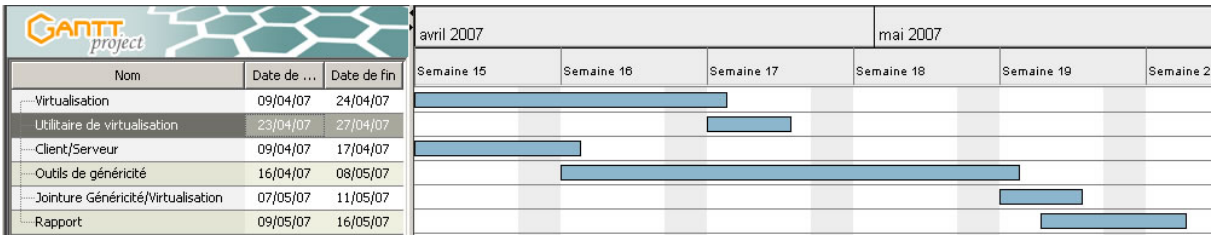

Figure 3. Petit diagramme de Gantt.

On note que des tâches de jointures ont été réalisées en parallèle à cause de notre modèle de développement incrémental. Ces tâches ne figurent pas dans le diagramme de Gantt[24] car elles sont considérées comme négligeables.

[24] Outil de gestion de projet

4. Développement du projet

Maintenant, détaillons la partie développement de notre projet. Elle se décompose en trois parties équilibrées.

4.1. Virtualisation

La virtualisation est la partie avant de notre système. C'est grâce à elle que le serveur frontal Java peut interagir avec la base de données Oracle. Ce module nécessite peu de code écrit, la majeure partie du travail étant la maîtrise d'une technologie préalablement choisie.

4.1.1. Principe

Nous utilisons un Framework[25] open source pour gérer la persistance des objets dans la base de données relationnelle Oracle. Ce Framework est Hibernate. C'est une bibliothèque écrite en Java qui propose un service de mapping[26] objet/relationnel. Nous pouvons nous baser sur la qualité et l'optimisation des requêtes SQL de ce Framework.
Durant la 1ère phase du TER (phase de recherche), nous avons longuement étudié ces librairies pour assurer que cet outil correspondait à nos besoins. Pendant la 2ème phase du TER (phase de développement à plein temps), nous avons testé et utilisé ce Framework pour en maîtriser le fonctionnement et l'appliquer à nos besoins.

Notre virtualisation de décompose en 3 parties :

- **Exploration et extraction du schéma de la base**
- **Configuration personnalisée du schéma**
- **Production du modèle objet**

4.1.1.1. Exploration et extraction du schéma de la base

Cette partie consiste à utiliser Hibernate pour explorer la base de données afin d'en extraire le schéma sous format XML. Deux outils Hibernate sont utilisables pour effectuer cette tâche : Hibernate Synchroniser et Hibernate Tools. Notre choix s'est porté sur Hibernate Tools qui est plus flexible que le Synchroniser.
Avant de lancer l'exportation, il est impératif de définir le fichier de configuration Hibernate qui comprend en outre les informations pour accéder à la base.
La base de données Oracle utilise 811 tables pour stocker toutes les informations utiles à la gestion d'une entreprise. L'extraction du schéma en XML est donc une étape longue. De par cette complexité, nous obtenons autant de schémas que de tables dans la Base.

Ci-dessous, figure 4, un aperçu de l'étape d'extraction. Elle correspond à une étape intermédiaire, dite de « filtrage », car il n'est ni pertinent ni utile d'extraire le schéma des autres NameSpaces[27] existants (Ceux liés au fonctionnement d'Oracle par exemple).

[25] Ensemble de bibliothèques permettant le développement rapide d'applications
[26] Procédé de virtualisation par association des données avec des tables de correspondances
[27] Espace de noms

Figure 4. Filtrage des Tables.

Ci-dessous, figure 5, un aperçu du résultat de l'extraction, sachant qu'il y a 811 fichiers dans ce répertoire.

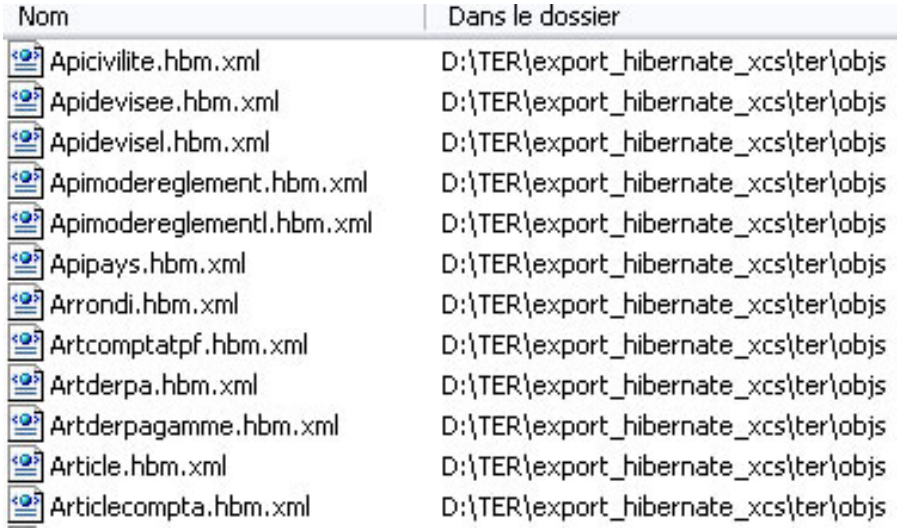

Figure 5. Résultat de l'extraction du Schéma.

L'étape d'extraction est donc terminée, donc nous pouvons passer à la phase de configuration. Il faut noter que cette étape doit être répétée à chaque installation de notre système, ou si la version de la base de données a changée. Elle est complexe, mais elle s'adresse à l'administrateur, qui est une personne avertie en informatique.

4.1.1.2. Configuration personnalisée du schéma

A partir des fichiers obtenus dans l'étape précédente, nous sommes en mesure d'appliquer notre utilitaire de configuration basé sur DOM XML. Cet outil récupère tous les fichiers de schémas dans un répertoire, et propose une interface intuitive pour effectuer des modifications. C'est en quelque sorte un éditeur pour l'XML, mais orienté fichiers de mapping pour Hibernate.

Le code écrit est sans surprise car il se base sur des éléments de cours vu au 1er semestre.

Ci-dessous, figure 6, une impression d'écran de notre outil de configuration des schémas de mapping. Cependant, cet outil n'est pas tout à fait fonctionnel.

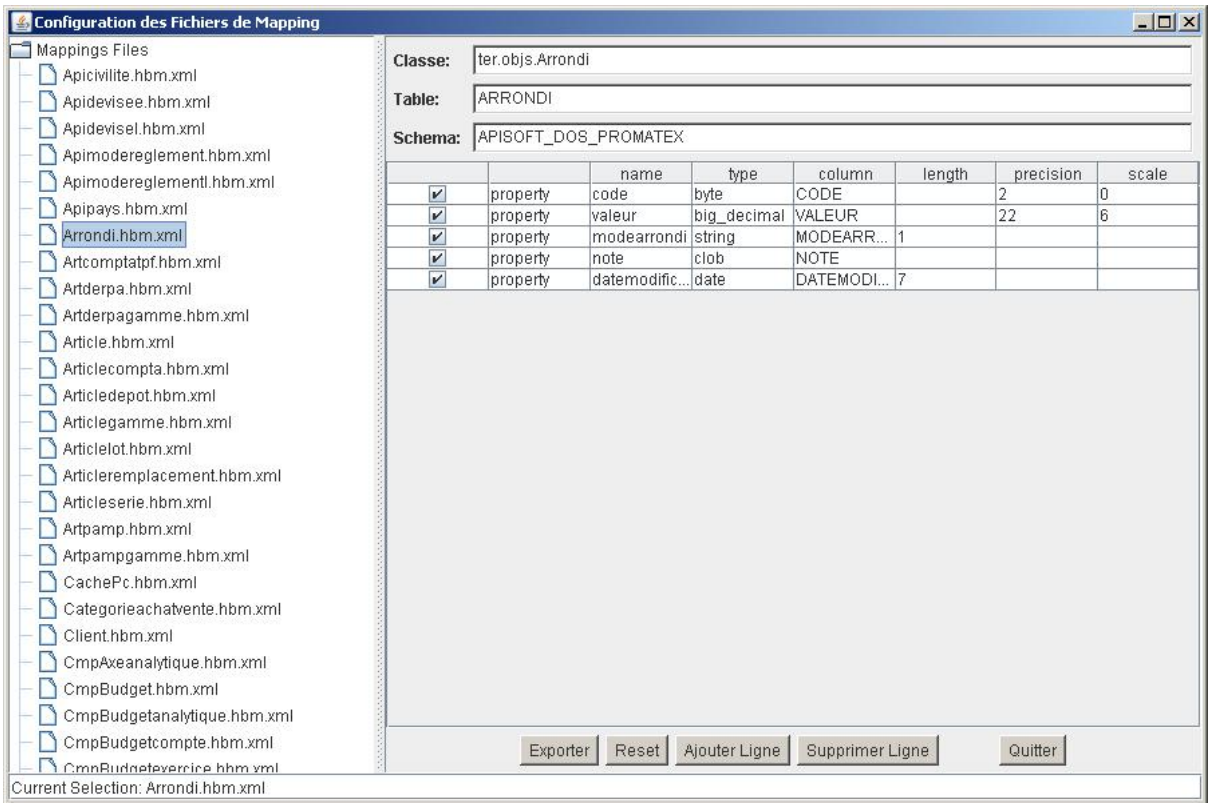

Figure 6. Utilitaire de configuration.

Nous pouvons alors exporter l'ensemble des schémas modifiés ou non vers un autre répertoire pour passer a l'étape suivante.

4.1.1.3. Production du modèle objet

Cette partie traite de la méthode dite « reverse engineering[28] ». Cela consiste, vis-à-vis de nos besoins, à générer automatiquement le code objet Java à partir des schémas XML configurés.

Cette génération de code Java est automatique, seul un fichier de configuration pour Hibernate est nécessaire. Ci-dessous, figure 7, une capture d'écran juste avant de lancer la procédure de « reverse engineering ».

[28] Rétro-ingénierie ou rétro-conception, méthode qui consiste à étudier un objet pour en déterminer le comportement

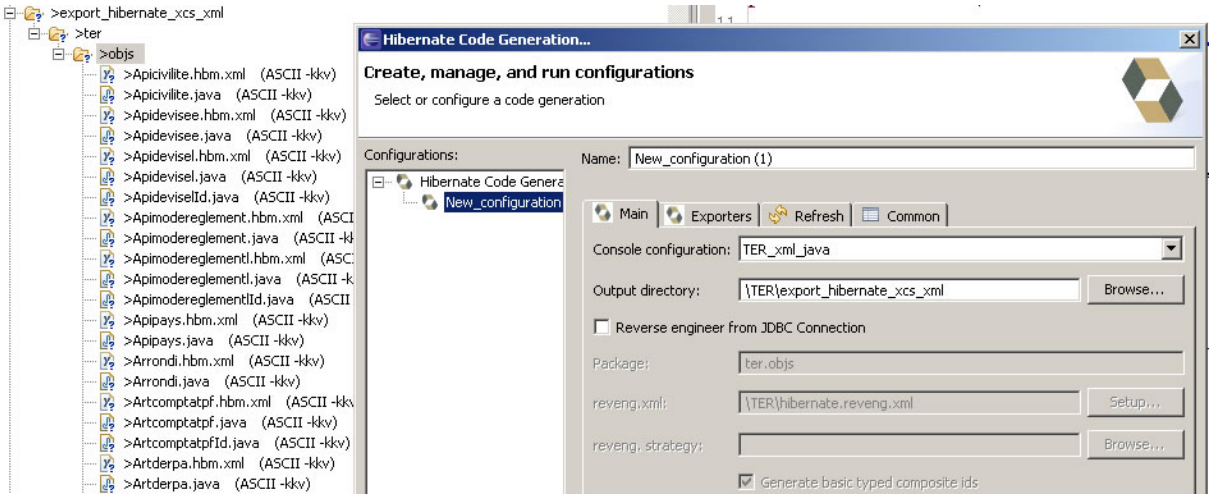

Figure 7. Génération des objets Java par la méthode « reverse engineering ».

Après la génération de code, nous obtenons autant de classes Java que de schémas de tables. Voici donc ci-dessous, figure 8, un aperçu du résultat. Nous ne vous présenterons pas le détail du code, car il est généré automatiquement pas Hibernate.

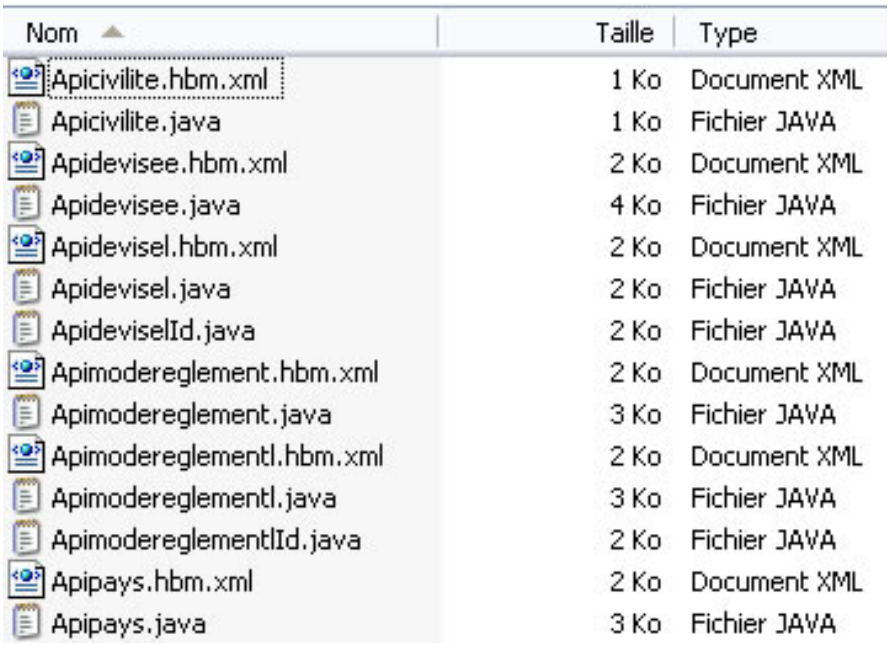

Figure 8. Fichiers Java générés.

Il reste à joindre la virtualisation au reste du projet. Délicate partie que nous avons mené avec succès.

4.1.2. Jointure dans le projet

La jointure avec le reste du projet est réalisée par une seule classe : Database.java. Cette classe est la seule qui contient du code non générique. C'est elle qui définit les points d'entrée de l'application Client/Serveur. Elle est composée des éléments suivants :

- Méthode static INIT : elle initialise la session Hibernate et démarre une transaction avec la base de données.

- Liste d'attributs static qui sont les Objets Java principaux de l'application commerciale (Liste de Clients, liste d'Articles, Liste de factures…). Ce sont les points d'entrées de l'application, le seul code en « dur » de notre système.
- Elle s'occupe de rendre les objets nouvellement créés ou modifiés persistants.

Ceci termine la partie avant de notre système. Cependant, voici quelques éléments qui nous ont pris plus de temps que prévus, et qui n'étaient pas triviaux à résoudre.

4.1.3. Problèmes et solutions

Certains éléments de la virtualisation nous ont posé des problèmes plus importants que d'autres.

Quelle technologie pour virtualiser une base de données Oracle ?
Nous sommes en présence de la concurrence de deux technologies différentes, Hibernate et JPA[29]. Notre choix s'est porté sur Hibernate qui semblait plus approprié à nos besoins. Ce choix s'est fait avant la pré-soutenance. Il s'est avéré justifié.

Comment rendre le mapping configurable ?
Pour répondre à ce problème, nous avons du créer un utilitaire se basant sur les fichiers de description de schémas de base de données d'Hibernate. Cet Outil est utilisable sur n'importe quelle base de données.

Jointure : Database.java
La classe Database est une partie vraiment importante de notre projet. C'est une réponse au problème de jointure avec le modèle Client/Serveur. Elle fait office de charnière.

4.2. Client/Serveur

Cette partie concerne le développement des applications client/serveur. L'application serveur communique avec la base via Hibernate et s'occupe de la gestion des objets persistants. L'application client demande des services au serveur et réalise l'affichage et l'interaction avec l'utilisateur.

Plusieurs technologies sont disponibles pour la communication entre le client et le serveur selon l'environnement de programmation. Imposée par le cahier des charges, l'application Client/Serveur doit être développée en Java. Nous avons donc choisi d'utiliser RMI[30] pour la communication. En effet RMI permet de concevoir un modèle orienté objet et de communiquer par interfaces.

Le serveur frontal Java est relié à la base de données Oracle par un réseau isolé. La liaison physique est réalisée par un câble croisé et des cartes réseaux de 100Mo.

[29] Framework concurrent à Hibernate
[30] Remote method invocation

4.2.1. Principe

Nous sommes partis du projet d'ALR et nous l'avons adapté à nos propres besoins. Nous avons commencé par créer l'interface qui définit les services qui sont fournis par le serveur. Le serveur est un objet distant qui implémente cette interface. Le client n'est pas un objet distant, car le serveur il ne fait pas de call-back[31].

Pour assurer la sécurité de la communication, nous avons utilisé RMI-SSL qui est la version sécurisée de RMI. RMI-SSL utilise des sockets SSL[32] et permet d'authentifier le serveur et de sécuriser les données.

Réalisation de RMI en SSL:

- Création des certificats pour le client et le serveur avec keytool[33]
- Utilisation des sockets customisés pour communiquer sous SSL

Fonctionnement de l'application :

- Serveur
  - Paramétrage du JVM[34]
    - Fichier de police
    - Certificat
    - Mot de passe pour le certificat
  - Lancement de rmiregistry[35]
  - Création de l'instance du serveur
  - Enregistrement de l'instance dans le rmiregistry
- Client
  - Paramétrage du JVM
    - Fichier de police
    - Certificat
    - Mot de passe pour le certificat
  - Paramétrage d'application
    - L'adresse IP du serveur
  - « Lookup[36] » dans le rmiregistry du serveur et récupérer le référence vers l'objet distant

4.2.2. Partie métier

Nous aborderons ici les grandes lignes suivies lors de l'implémentation des services offerts par le serveur et les fonctionnalités fournies par l'application client aux utilisateurs.

4.2.2.1. Serveur

- Récupération des objets proxy
  - Récupérer les objets persistants via Hibernate

[31] Fonction de rappel
[32] Secure Socket Layer, protocole de sécurisation des échanges
[33] Outil fourni avec le jdk de Java pour créer et signer des certificats
[34] Machine Virtuelle Java
[35] Gestionnaire de service de nomage RMI
[36] Rechercher

    - Créer l'objet distant qui fait proxy pour chacun des objets persistants
- Ajout d'un objet
    - Créer l'objet par l'introspection
    - Rendre l'objet persistant
- Suppression d'un objet
    - Récupérer l'objet persistant
    - Supprimer l'objet de la base
    - Détruire l'objet
- Commit
    - Valider la transaction en cours. Ce processus enregistre toutes les modifications qui ont été faites sur les objets persistants et vide le conteneur de gestion. Les objets persistants deviennent volatiles.
    - Recréer une transaction
    - Remettre tous les objets dans le conteneur de gestion

4.2.2.2. Client

- Affichage des objets
    - Récupérer une liste d'objets distants de même type
    - Analyser l'objet
    - Afficher dans l'interface graphique par l'introspection
- Modification d'un objet
    - Récupérer l'objet à modifier
    - Récupérer l'attribut à modifier
    - Récupérer la nouvelle valeur
    - Récupérer la méthode modifier
    - Exécuter la méthode en passant en paramètre la nouvelle valeur.
- Impression de document
  Nous avons utilisé Java Print Service pour faire l'impression. Nous avons créée la classe ImprimerTable qui implémente l'interface Printable. Cette classe définit une méthode print qui fait l'impression.
    - Récupérer la liste des objets affichés
    - Les mettre dans un tableau
    - Créer une instance de la classe ImprimerTable en lui passant en paramètre le tableau.
    - Appeler la méthode print sur cette instance.
- Génération de PDF
  Nous avons utilisé l'outil libre iText pour créer un document en PDF. Pour cela, nous avons créé la classe GenerateurPDF qui utilise cette librairie et définie une méthode genererPDF.
    - Paramétrage d'application
        - Le nom du fichier à générer
        - La liste d'objet
    - Créer un objet pdf
    - Créer un objet de tableau
    - Initialiser le tableau avec les éléments de la liste
    - Ajouter le tableau dans l'objet pdf
    - Créer le fichier et sérialiser l'objet dans le fichier

Notre application peut être utilisée depuis l'internet grâce à la technologie Java Web Start. L'utilisateur pourra travailler avec le logiciel client sans aucune installation. Ci-dessous, figure 9, le site de lancement.

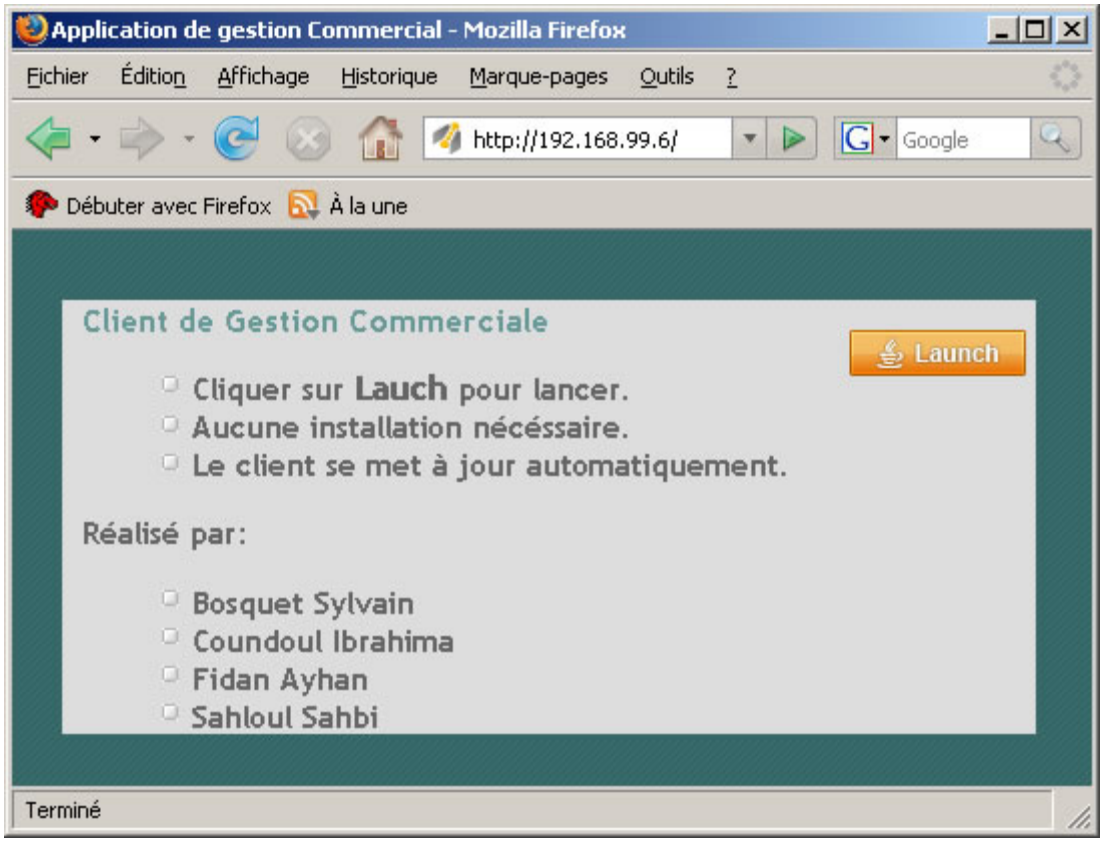

Figure 9. Java Web Start.

4.2.3. Partie Test

Dans le projet, nous avons effectué des tests unitaires avec JUnit pour vérifier le bon fonctionnement des modules. Les tests réalisés :

- Affichage des objets distant

Les objets distants représentent la base et font des proxy sur des objets persistants. Nous avons testé la correspondance des valeurs des objets avec la base.

- Modification d'un objet

Nous avons vérifié si la modification d'un objet distant sera reflétée dans la base.

- Ajout d'un objet

Après la création d'un objet, nous l'avons rendu persistant et avons testé s'il existe dans la base.

- Suppression d'un objet

Nous avons testé si un objet existe dans la base après sa suppression.

4.2.4. Problèmes et solutions

Les trois problèmes ci-dessous ont été rencontrés. Le premier est lié aux technologies utilisées (RMI), tandis que les deux autres sont liés aux codages.

- Configuration des machines et des réseaux

Pour le fonctionnement de RMI, il faut assurer que les ports utilisés par RMI sont ouverts. Nous avons configuré le routeur et le pare-feu sur des machines des clients.

- Commit

Après chaque commit, Hibernate enregistre les objets persistants dans la base et termine la transaction. Hibernate vide le conteneur de gestion et les objets persistants deviennent volatiles. Donc les modifications qui sont faites sur des objets volatiles ne seront pas enregistrées dans la base lors de la prochaine commit. Pour résoudre le problème, nous gardons la liste des objets persistants. Après le commit, nous créons une nouvelle transaction et rendons les objets volatiles persistants.

- Initialisation des instances

Quand nous avons créé des instances à partir des classes générées par Hibernate, l'interface graphique lance « NullPointerException » car les attributs ne sont pas initialisés. Pour résoudre ce problème, nous initialisons les attributs de type :

  - Int avec 0
  - Bigdecimal avec 0
  - Double avec 0.0
  - String avec une chaîne vide
  - Date avec la date actuelle

4.3. Outils de généricité

Dans notre projet, nous nous sommes aperçus que la récupération d'un objet persistant est un processus lourd, si l'objet contient beaucoup d'associations. Pour optimiser la performance, nous avons pensé à utiliser le mode de récupération paresseuse de Hibernate (lazy loading). Nous ne faisons pas transiter sur le réseau les objets pour économiser la bande passante, mais nous créons des objets distants qui feront office de proxy sur les objets persistants. Nous récupérons les associations d'objets à la demande.

4.3.1. Principe

La génération des classes proxy se fait en deux étapes :

- **L'Analyse de la classe**

En premier, nous avons utilisé l'introspection de Java pour analyser une classe. Nous avons créé une classe qui s'occupe de cette tâche. Toutes les classes qui sont générées par Hibernate contiennent des attributs privés, des méthodes accesseurs et des méthodes modifieurs. Notre classe « ObjectAnalyser » récupère les méthodes accesseurs, les méthodes modifieurs et les attributs à partir des méthodes accesseurs. Nous pourrons exécuter une méthode grâce à la réflexivité en Java. Cette classe implémente une méthode aussi pour indiquer si un attribut est complexe (pas de type primitif, ni de type String ni de type Date).

- **Génération de la classe distante et de l'interface distante**

En deuxième et dernière étape, nous générons les classes distantes et interfaces distantes. Nous avons créé la classe « RemoteClasseGenerator » pour les créer à partir d'une classe générée par Hibernate.

- Génération de l'interface distante
  - Analyser la classe
  - Etendre l'interface java.rmi.Remote
  - Générer les mêmes méthodes que la classe
  - Chaque méthode lance l'exception java.rmi.RemoteException
- Génération de la classe distante
  - Etendre la classe java.rmi.server.UnicastRemoteObject
  - Implémenter l'interface distante générée
  - Englober un attribut du même type que la classe
  - Implémenter les méthodes définies dans l'interface
    - Pour les méthodes accesseurs
      - Exécuter la même méthode accesseur de l'objet englobé
      - Retourner le même objet.
    - Pour les méthodes modifieurs
      - Exécuter la même méthode modifieur de l'objet englobé en passant le même paramètre

4.3.2. Problèmes et solutions

Dans cette section, nous avons rencontré des problèmes avec la gestion des attributs complexes. Un attribut est complexe s'il n'est pas de type primitif, ni de type String, ni de type Date. Nous avons distingué deux cas pour gérer les attributs complexes :

- L'attribut complexe est de type collection. Cela veut dire qu'il appartient au package « java.util » et contient une liste d'objets éventuellement du même type. C'est plutôt l'attribut qui représente une association 1-n ou n-m. Dans ce cas, il faut modifier c'est les éléments de la liste.
- Le type d'attribut complexe est généré par Hibernate. L'attribut représente plutôt une association 1-1 ou n-1. Dans ce cas, il faut analyser l'attribut récursivement.

5. Etat d'avancement du projet

A la date du 16 Mai 2007, alors que le rapport est prêt à être rendu, voici où nous en sommes de notre développement.

La partie avant de notre système, (la virtualisation), est opérationnelle. Il est par contre intéressant de préciser que l'extraction du schéma de la base est une étape longue. Mais elle fonctionne parfaitement. Seule ombre au tableau, l'utilitaire de configuration développé par Sahbi ne fonctionne pas.
La partie arrière, (le modèle Client/Serveur et les Outils de généricité), est également opérationnelle. Le client se lance par le biais d'un site internet avec les outils Java Web Start.

Pour résumer, le serveur sait communiquer avec la base de données Oracle de manière générique, et le client affiche les résultats également de manière générique. C'est donc un système minimal. Notre modèle incrémental a fonctionné et nous possédons le cœur du système de virtualisation demandé.

Cependant, vu l'abondance du travail et le temps limité, nous avons du nous arrêter à une boucle incrémentale inférieure à celle souhaité.
En effet, l'interface graphique du client est sommaire. Elle ne sait pas gérer les pièces commerciales (devis, factures, bons de commandes…). Nous n'avons pas pu aborder cette partie et nous le regrettons. Elle aurait pu nous ouvrir une porte sur la gestion d'entreprise, elle aurait pu ajouter à notre bagage informatique des notions précieuses de gestion, agréablement vu dans les entreprises.

6. Conclusions et perspectives

Force est de constater que le TER est un travail enrichissant. Il permet de compléter nos connaissances dans des domaines pointus que nous n'aurions pas abordés en cours. Il nous introduit également aux conditions de travail que nous trouverons à la fin de nos études. Nous avons appris à travailler avec des personnes que nous ne connaissons pas, et nous avons du faire face à certains conflits.

Nous regrettons sincèrement de ne pas avoir pu satisfaire entièrement le cahier des charges, seulement à cause du temps, pas à cause de nos compétences. Nous avons exploité au mieux notre temps en partageant les tâches et en se concertant sur toutes les décisions importantes. Grâce à notre modèle de développement incrémental, nous avons pu présenter un système fonctionnel à notre encadrant.

Si dans un avenir proche, un budget était alloué pour la continuité de ce projet, nous sommes fier de laisser à nos successeurs un système fonctionnel et performant, bâtit sur une étude approfondie. Dans cette hypothèse, nous sommes prêts à accompagner l'équipe qui prendra le relais par nos modestes connaissances.
Cependant, en espérant que l'entreprise SICA soit satisfaite de notre travail, nous serions heureux d'accepter toutes formes de stages ou autres contrats pour continuer l'aventure que nous à fait découvrir notre TER.

7. Références

Hibernate :

- Cours de Base de données Avancées Master 1 Informatique 2005-2006
  M. Richard Grin
  http://deptinfo.unice.fr/~grin/anciensCours/2005-06/minfo/bdavancees/

- Site de Référence pour Hibernate
  http://www.hibernate.org

- La documentation d'Hibernate Tools
  http://www.hibernate.org/hib_docs/tools/reference/en/html/

- Wikipédia en version française
  http://fr.wikipedia.org

Java Web Start :

- Java Web Start par SUN
  http://java.sun.com/products/javawebstart/

DOM :

- Cours de programmation Web Master 1 Informatique 2006-2007
  M. Philippe Poulard
  http://disc.inria.fr/perso/philippe.poulard/cours/master/

Client/Serveur et Généricité :

- Cours d'ALR de Master 1 d'Informatique de l'UNSA 2006-2007
  M. Denis Caromel
  http://www.inria.fr/oasis/Denis.Caromel/ProgRpt/

- Cours de Génie Logiciel Orienté Objet 2006-2007
  M. Philippe Collet
  http://deptinfo.unice.fr/twiki/bin/view/Minfo/GLOO

- Documentation de Sun
  http://java.sun.com/docs/books/tutorial/uiswing/components/table.html
  http://java.sun.com/j2se/1.4.2/docs/guide/jps/index.html
  http://java.sun.com/javase/6/docs/technotes/guides/rmi/index.html
  http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/samples/
  http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html
  http://java.sun.com/docs/books/tutorial/uiswing/components/table.html

8. Annexes

8.1. Définitions et acronymes

Virtualisation :	Système logiciel d'abstraction du fonctionnement d'une base de données sous Oracle.
Mapping :	Procédé de virtualisation par association des données avec des tables de correspondances.
Front office :	Gestion des accès, guichet.
Portable :	Non dépendant du système d'exploitation.
Isolation galvanique :	Isolation totale.
Introspection :	Analyse de soi.
Hibernate :	Framework open source gérant la persistance des objets en base de données relationnelle.
Généricité :	Indépendance vis-à-vis du type d'objet manipulé.
Wiki :	Système de gestion de contenu de site Web.
Horde :	Logiciel open source pour la gestion des e-mails et des calendriers.
CVS :	Logiciel libre de gestion de versions (Concurrent Versions System).
Eclipse :	IDE écrit en java permettant de créer des projets de développement dans n'importe quel langage.
IDE :	Environnement de développement intégré.
JBOSS IDE ECLISPE :	Version modifiée d'Eclipse optimisée pour Hibernate.
Ubuntu :	Distribution Linux basée sur debian.
Code métier :	Savoir faire du serveur d'application.
DOM :	Document Object Model.
XML :	Extensible Markup Language.
Gantt :	Outil de gestion de projet.
Framework :	Ensemble de bibliothèques permettant le développement rapide d'applications.
Call-back :	Fonction de rappel.
RMI :	Remote method invocation.
NameSpace :	Espace de noms.
SSL :	Secure Socket Layer, protocole de sécurisation des échanges.
Keytool :	Outil fourni avec le jdk de Java pour créer et signer des certificats
Reverse engineering:	Rétro-ingénierie ou rétro-conception, méthode qui consiste à étudier un objet pour en déterminer le comportement.
JPA :	Framework concurrent à Hibernate.
JVM :	Machine Virtuelle Java.
Lookup :	Rechercher.
Rmiregistry :	Gestionnaire de service de nomage RMI
Gestion commerciale :	Gestion de pièces commerciales.
Pièces commerciales :	Devis, factures, bon de commandes…
Gestion de fichiers :	Gestion des Articles, clients, fournisseurs ...