République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université d'Ibn Khaldoun – Tiaret

Faculté des Mathématiques et de l'Informatique

Département Informatique

Thème

Conception et Réalisation d'une application distribuée

« Agence Touristique »

Pour l'obtention du diplôme de Master II

Spécialité : Réseaux et Telecom

Rédigé par : Safir Fatima
Mimouni Zoubeyda

Dirigé par : Mr. MOSTEFAOUI Sid Ahmed

Année universitaire : 2015-2016

Remerciement

Nous remercions en premier lieu ALLAH qui nous a éclairé le chemin du savoir et qui nous a donné la volonté et la patience d'achever ce modeste travail de mémoire, notre grand salut sur le premier éducateur notre prophète Mohamed (satisfaction et salut de dieu soit sur lui).

Nous tiens à adresser mes remerciements aux nos parents qui a consenti des sacrifices et prodigué des encouragements tout au long de mes études.

Nous adressons nos vifs remerciements et nos sincères gratitudes, à notre Encadreur Monsieur ***MOSTEFAOUI SID AHMED*** *qui nous a fait l'honneur d'avoir la charge d'encadrer notre travail de mémoire avec grande patience, pour la confiance qu'il a eu en notre projet et surtout pour ses orientations, ainsi que son aide précieuse et ses conseils pour réaliser cette mémoire.*

Nous Remercions également notre jury d'avoir accepté de juger notre travail et Nous tenons à remercier également nos collègues pour leur aide de réalisation ce Modeste Mémoire.

Je dédie cette thèse à…

Mes labours à la source de tendresse de générosité :

A la lumière de ma vie et l'espoir de mon existence…ma mère Radia.

A mon défunt père que Dieu bénisse.

A mes chers frères et sœurs.

A tous les membres de ma famille

A mon binôme Mimouni Zoubeyda

A tous mes amis sans exception

A ma promotion d'informatique générale

Et à tous les professeurs qui m'ont aidée tout au long de mes études.

Sans oublier tous ceux qui ont par leurs égard contribué à parfaire mon objectif

et qui me font l'éminent honneur avec différence, je tien à leur dédier ce travail.

Safir Fatima

Je dédie cette thèse à…

A ma très chère mère Affable, honorable, aimable : Tu représentes pour moi le symbole de la bonté par excellence, la source de tendresse et l'exemple du dévouement qui n'a pas cessé de m'encourager et de prier pour moi.

A mon Père Aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que j'ai toujours eu pour vous. Rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien être.
Ce travail est le fruit de tes sacrifices que tu as consentis pour mon éducation et ma formation

A Mes grandes mères zoubida, khadidja.
A la mémoire de mes grands-pères.

A Mon très chère frère Med Hamani
A Mes chères sœurs khadouj, Amouna.

A Mes Chers Tantes Meriem, Zohra, Anissa, Kheira, Hafsa, Salima, Fairouz, Amina
A Mes Chers Oncle Mustapha, Salim, Sid Ali, Faisal, Belkacem
A Mes Chers Cousins et Cousines : Nawel, Fatima, Amina, Loubna, Zouzou, Amouna, Khadouj, Afa, et Douaa
A Les enfants de ma famille Yassine, Nadir, Abderrahmane, Adem et Yasser

A mes chères amies : Zouzou, Imoucha et son marie Hamada, Djamila, Nahla
A mon binôme Safir Fatima
A tous mes collègues du travail

Pour ceux qui je n'ai pas cité bien sûr ne croyait pas que je vous ai oublié, je vous porte toujours dans mon cœur.

Mimouni Zoubeyda

Résumé

Notre travail a pour objectif de développer une application répartie dans le modèle distribué, consistant à informatiser la gestion d'une agence de voyage. L'utilité de cette informatisation est de faciliter la gestion de l'agence pour l'administrateur et garder un contact permanent avec le client. Pour cela nous avons choisi la technologie de composants entreprise Java Bean (EJB) comme un outil de développement.

Ces composants préservent les avantages de la programmation orienté-objet tout en adressant les activités complexes de déploiement et de contrôle à l'exécution des applications. Les EJBs qui sont des composants java portables permettant, une fois assemblés entre eux, de créer des applications s'exécutent dans un conteneur EJB (environnement d'exécution des EJB) qui leur fournit de nombreux services : la transaction, la sécurité, la localisation et la persistance.

Mots-clés : application répartie, distribuée, programmation orienté-objet, EJB.

Table des matières

Introduction générale 1
Chapitre 01 : La programmation distribuée 3
Introduction : 3
I. Historique : 3
II. Système centralisé : 4
II.1. Applications monolithiques : 4
II.2. Applications client/serveur : 4
II.2.1. Architecture de modèle Client/serveur : 4
II.2.2. Architecture centralisée à serveurs multiples : 5
II.2.3. Les inconvénients : 5
III. Système distribué : 5
III.1. Définition 1 : 5
III.2. Définition 2 : 6
III.3. Architecture distribuée : 6
IV. Qu'est-ce que le middleware : 7
IV.1. Les objectifs principaux du middleware : 7
IV.2. Les services des middlewares : 7
IV.3. Nombreux middlewares : 7
V. Les différentes architectures (serveur/client) : 8
V.1. L'architecture 3 tiers : 8
V.1.1. Intérêt potentiel : 8
V.2. L'architecture n-tiers : 9
VI. Les composants de système distribué : 9
VII. Les objectives du système distribue : 10
VIII. Domaines et exemples d'application : 10

VIII.1. Le Peer-to-Peer (poste à poste) : 11
VIII.2. Web (http) : 11
IX. Intérêt fonctionnels : 11
Conclusion : 12
Chapitre 02 : les technologies de programmation distribuée 13
Introduction : 13
I. RMI : 13
I.1. Définition : 13
I.2. Architecture de RMI : 13
I.3. Avantages et inconvénients de RMI : 15
Il. CORBA : 16
Il.1. Définition : 16
Il.2. Principe de CORBA : 17
Il.3. Architecture CORBA : 17
Il.4. les Avantages et les Inconvénients de CORBA : 18
Il.5.Objectifs : 19
III. .NET : 19
III.1. Définition : 19
III.2. Le Framework .NET : 20
III.3. Objectifs de .NET : 21
III.4. Principales caractéristiques de .NET : 21
III.4.1. Interopérabilité : 21
III.4.2. Common Runtime Engine : 22
III.4.3. Indépendance du langage : 22
III.5. Structure d'une application .NET : 22
III.6. Avantage : 22
IV. J2EE : 22

IV. 1. Définition : 22
IV.2. Architecture : 23
IV.2.1. Couche présentation (client) : 24
IV.2.2.Couche application : 24
IV.2.3. Couche métier, ou logique business : 24
IV.3. Les composants JEE : 25
IV.3.1. JPA : 25
IV.3.2 Servlet : 25
IV.3.3. Pages JSP : 25
IV.3.4. JSF : 25
IV.3.5. EJB 3.0 : 25
V. EJB : 26
V.1. Définition : 26
V.2. L'architecture EJB : 26
V.3. La présentation des EJB : 28
V.4. Caractéristiques d'EJB : 29
V.5. Les principes : 30
V.6. Composants entreprise java beans : 30
V.7. visibilité des EJB : 30
V.7.1. visibilité locale : 31
V.7.2. visibilité distante : 31
V.7.3. visibilité service web : 31
V.8. les diffèrent type d'EJB : 32
V.9. Les type de Bean : 32
V.9.1. les EJB session : 32
V.9.2. Les EJB entité : 33
V.9.3. Les EJB message : 34

V.10. Création d'un EJB : 34
V.10.1. conteneur d'EJB : 34
V.10.2. serveur EJB : 34
V.10.3. java Naming and Directory Interface JNDI : 34
V.11. les composants d'un EJB : 35
V.11.1. Entreprise Bean : 35
V.11.2. Interface Home : 35
V.11.3. Interface Remote : 35
V.11.4. Descripteur de Déploiement : 35
V.11.5. Fichier EJB-Jar : 35
V.12. EJB 2.0 : 36
V.12.1. session Bean : 36
V.12.1.1. Méthodes du cycle de vie : 36
V.12.1.2. Les interface : 37
V.12.1.3. Le descripteur de déploiement : 37
V.12.2. Entity bean CMP : 37
V.12.2.1. La classe du Bean : 38
V.12.2.2. Message Driven Bean : 39
V.13. Les inconvénients des EJBs 2 : 39
V.14. EJB 3.0 : 40
V.14.1 Session bean: 41
V.14.1.1 La classe du Bean: 41
V.14.1.2 Les méthodes du cycle de vie : 41
V.14.1.3. Les interface métiers : 43
V.14.2. Entity Bean : 43
V.14.2.1. La classe de l'entité : 44
V.14.3. Message driven bean : 45

V.14.3.1. La classe du Bean : 45
V.14.3.2. Message Driven Context : 45
V.15. EJB pour développer des composants business : 45
V.16. les avantages : 46
Conclusion : 46
Chapitre 03 : Conception et Réalisation 47
Introduction : 47
I. Système d'information 47
I .1.Définition : 47
I .2.Les phases du SI (cycle de vie) : 47
I .2.1.Recueil des besoins : 47
I .2.2.Conception : 47
I .2.3.Exploitation : 47
I .2.4.Production : 48
I.3.Modélisation de SI : 48
II. La méthode MERISE : 48
II.1. Définition : 48
II.2. Le modèle conceptuelle des données (MCD) : 49
II.2.1. Les modèles schématiques de représentation : 49
II.2.2. MCD vérifier et normalisé : 50
II.3.Le modèle conceptuel des traitements (MCT) : 50
II.3.1. Représentation du MCT : 51
II.3.1.1. Partie client : 51
II.3.1.2. Partie administrateur : 52
II.4. Le modèle logique des données(MLD) : 52
II.4.1. Élaboration du MLD : 53
III. Langage utilisé : JAVA 53

III.1. Définition : 53
III.2. L'avantage du langage java : 53
IV.1. Les outils utilises : 54
IV.1.1. MySQL 5.0 : 54
IV.1.2. NetBeans 7.2.1 : 54
IV.1.3.GlassFish 3.1.2 : 54
V. Les étapes de la réalisation du projet : 54
V.1. Création d'une base de données : 55
V.2.Création des composants EJB sous NetBeans : 55
V.2.1.Création d'une class Library : 56
V.2.2.Création d'un module EJB : 57
V.2.3. Création des composants sessions pour chaque Entity Bean : 59
V.2.4.Déploiement de l'EJB : 61
VI. Application cliente : 62
VII. Les Interfaces graphiques : 63
VII.1.Iinterface générale : 63
VII.2. Coté client : 63
VII.2.1. Réservation : 63
VII.2.2.Nos destination : 64
VII.2.3.Contacte : 65
VII.2.4.Consultation : 65
VII.2.5.Aide : 66
V.III. Coté administrateur : 66
V.III.1.Login : 66
V.III.2. Interface de l'administrateur 67
V.III.2.1. Mise à jour des voyages : 67
V.III.2.2. Recherche 68

V.III.2.3. Mise à jour des clients : 69
V.III.2.4.Afficher : 69
Conclusion : 69
Conclusion générale : 71

Liste des figures :

Chapitre 01 : La programmation distribuée

Figure 01 : Le modèle client/serveur…….....4
Figure 02 : L'architecture distribuée……..6
Figure 03 : L'architecture 3 tiers……...8
Figure 04 : L'architecture n-tiers…….....9

Chapitre 02 : Les technologies de programmation distribuée

Figure 05 : Processus de fonctionnement de RMI…….……..13
Figure 06 : Architecture de RMI……..14
Figure 07 : Notion client/serveur avec le bus de CORBA……...17
Figure 08: Architecture CORBA…….18
Figure 09: L'architecture de .NET……...20
Figure 10: L'architecture de .NET Framework…….……..21
Figure 11: L'architecture J2EE……23
Figure 12: Architecture EJB simple…….27
Figure 13: Architecture EJB type business to business……...27
Figure 14 : Serveur EJB…….....28
Figure 15 : L'exécution d'EJB……29
Figure16 : Cycle de vie d'une session Bean…….....37
Figure17 : Cycle de vie d'un Entity Bean……38
Figure 18 : Cycle de vie d'un Stateless Session Bean (EJB 3)……42
Figure 19 : Cycle de vie d'un Stateful Session Bean (EJB 3)……...……..42

Chapitre 03 : Conception et Réalisation

Figure 20 : Modèle conceptuel de données (MCD)…….……....50
Figure 21 : Le modèle conceptuel des traitements (MCT client)……...…….51
Figure 22 : Le modèle conceptuel des traitements (MCT admin)……..….....52
Figure 23 : Le modèle logique des données(MLD)…….……53
Figure 24 : La base de données……..…….……..55
Figure 25 : Schéma général de l'application…….…...56
Figure 26 : Création de nouveau projet "Java Class Library"…….....56

Figure 27 : Création de la connexion et l'ajout d'entité Bean pour chaque table…….....57
Figure 28 : Création d'un EJB Module……57
Figure 29 : Création d'un nouveau projet (Nom et place du projet)……58
Figure 30 : Création d'un nouveau projet (choix du serveur d'application)…….…...58
Figure 31 : Création d'une session beans Entity classe…….…..59
Figure 32 : Création de l'unité de persistance……..60
Figure 33 : Choix du fournisseur de la persistance…….….60
Figure 34 : Fichier de configuration……61
Figure 35 : Configuration de l'unité de persistance…….....61
Figure 36 : Déploiement de projet EJBAgence……...62
Figure 37 : L'interface de l'application……...63
Figure 38 : Réservation de voyage..…….……64
Figure 39 : Destination de l'agence…….……64
Figure 40 : Contacte…….……65
Figure 41 : Consultation des voyages……...…...…65
Figure 42 : Aide…….....66
Figure 43 : Login de l'administrateur……..67
Figure 44 : Mise à jour des voyages……67
Figure 45 : Fenêtre de la recherche……..……68
Figure 46 : Fenêtre de consultation administrateur……..68
Figure 47 : Mise à jour des clients……..69
Figure 48: Affichage des clients……..69

Liste des tableaux :

Tableau 1 : Les niveaux d'abstraction de merise……49

Introduction générale

L'architecture logicielle est aujourd'hui une partie primordiale du développement d'applications. Bien que complexe, le développement en tiers et en couches est un passage obligatoire pour tout développement d'applications orientées entreprise. Il permet de diviser un logiciel en parties plus petites pour en maîtriser la complexité.

La complexité des systèmes à prendre en compte n'a cessé de croître. De nombreux outils, méthodes, et architectures ont été proposés, pour maîtriser la conception, l'implémentation, le déploiement et la maintenance tout en respectant. Les spécifications et les échéances établies en amont. Dans cette évolution au fil des ans, on a vu apparaître le principe d'interegiciele, qui fournit une couche intermédiaire permettant d'abstraire un certain nombre de services pour simplifier la mise en œuvre du logiciel dans tous ses cycles de développement. Couche après couche.

L'approche du modèle d'architecture « distribuée » impose l'idée qu'une application est découpée en plusieurs unités. Cependant, il serait quasi impossible de créer ce type d'application (à partir de rien). En effet, la seule phase de création de la plate-forme supportant ce type d'applications aurait un coût important aussi bien en termes d'argent que de temps.

Ce projet s'intéresse donc à la manière de gérer efficacement de telles applications sur des environnements distribués. En partant d'approches existantes des applications distribuées, pour ça on a utilisé les nouvelles infrastructures d'application distribuée et ses services.

Ce mémoire est organisé comme suit :

***-Chapitre* 01 :** introduit de manière générale l'état de l'art de la programmation distribué en commençant par un bref historique du composant, puis la différence entre une application centralisée et une application distribuée. Notamment les définitions, les architectures, les inconvénients, les avantages et les buts de chaque applications, En passant en revue les modèles client /serveur, nous concluons avec les objectifs de l'application distribuée.

***-Chapitre 2* :** nous étudierons en détail les technologies de la programmation distribues par composant, en définissant les composants, le modèle RMI, l'architecture du CORBA avec les caractéristiques, les avantages et les inconvénients ,puis la technologie .NET, l'architecture J2EE enfin en parle sur les EJB et ses types.

-Chapitre 3 **:** Il illustre l'implémentation et la réalisation de l'application qui est une gestion d'agence touristique répartie à base composants EJB. On termine par une partie de tests d'exécutions de notre application.

Enfin, une conclusion synthétisera notre travail et présentera les perspectives envisagées.

Chapitre 01 : La programmation distribuée

Introduction :

Les besoins croissants des utilisateurs des systèmes et des logiciels, l'évolution rapide du matériel, l'explosion des réseaux informatiques ont motivé l'émergence de nouvelles approches de développement d'applications à grande échelle.

Le modèle d'architecture « distribuée » impose l'idée qu'une application est découpée en plusieurs unités. Cependant, il serait quasi impossible de créer ce type d'applications du néant.

En effet, la seule phase de création de la plate-forme supportant ce type d'applications aurait un coût important aussi bien en termes d'argent que de temps.

Nous détaillerons le point essentiel de ces technologies dites «les applications distribuées». Puis nous consacrerons plus de détails à l'implémentation.

I. Historique :

Les concepts d'architecture ont subi de nombreuses évolutions depuis les premiers développements d'applications.

Les premières applications étaient composées d'une seule « pièce», pour ne pas dire d'une seule fonction linéaire et totalement séquentielle. Si cette solution avait l'avantage d'être performante, elle avait aussi de nombreux inconvénients : évolution difficile, maintenance lourde, partage difficile des données…

Pour éviter ces désagréments, de nouveaux systèmes ont vu le jour. Les bases de données ont tout d'abord simplifié le partage et l'échange de données. L'arrivée d'Internet et des réseaux d'entreprise ont ensuite permis l'utilisation d'applications clientes plus génériques (les navigateurs web, par exemple). Leurs architectures ont dû ainsi évoluer, afin de séparer au mieux les différentes parties de ces applications.

On a alors assisté au succès du modèle *3-tiers* (3 parties) qui s'est finalement généralisé en un modèle *n-tiers*, découpant l'application en *n* parties.

L'industrialisation des développements et la taille grandissante des équipes ont aussi participé à l'évolution de ce nouveau modèle de découpage.

Les développeurs souhaitant accélérer leurs développements, ont dû factoriser leur travail pour ne pas avoir à toujours « tout recréer » à chaque projet. Ils ont, pour cela, développé des« briques » réutilisables et interconnectables facilement, réduisant, ainsi, le temps et le coût des développements.**[1]**

II. Système centralisé :

Ce système est basé sur un ou plusieurs serveurs qui possèdent la liste des fichiers partagés et qui orientent les internautes.

II.1. Applications monolithiques :

Une application monolithique est un programme constitué d'un seul bloc et s'exécutant sur une seule machine. Les applications monolithiques sont généralement utilisées dans le domaine du temps réel ou bien au sein d'applications demandant de grandes performances.

Elles restent également omniprésentes dans le monde du grand public, étant utilisées en standalone sur les machines personnelles.

II.2. Applications client/serveur :

Ces applications permettent de séparer la partie cliente et de regrouper la partie applicative sur un serveur. Cependant, le développement de ce genre d'applications nécessite la création d'un protocole de communication entre le client et le serveur. Ce protocole étant souvent propriétaire, l'évolution de ces applications doit souvent être faite par les créateurs.

II.2.1. Architecture de modèle Client/serveur :

Ce modèle consiste à créer un lien entre le client et serveur par l'intermédiaire d'une requête selon les principes suivants :

- Application cliente (interface homme/machine et traitements) située sur le poste client.
- Lien direct entre client et serveur.
- Traitements au niveau du serveur.

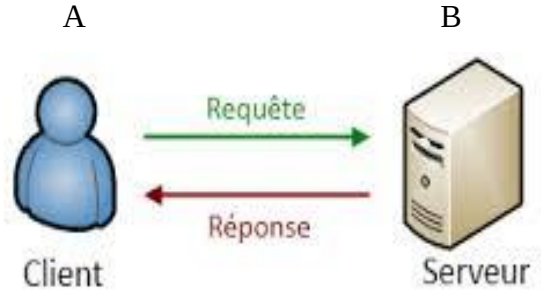

Figure 01 : Le modèle client/serveur

La Figure 01 illustre le processus d'architecture client/serveur. Un Client hébergé sur une machine A, envoie sa requête au processus Serveur, hébergé sur la machine B. Celui-ci renvoie à son tour le résultat de la requête.

II.2.2. Architecture centralisée à serveurs multiples :

Cette architecture est une amélioration de l'architecture centralisée, l'unique serveur est remplacé par un réseau de serveurs, les serveurs sont ensuite en mesure de se connecter entre eux, en fonction de leur connaissance les uns des autres. On peut donc avoir plusieurs réseaux indépendants. Le client choisit ensuite à quel serveur il souhaite se connecter

Le réseau plus connu utilisant cette architecture est **endon Key** [2]

II.2.3. Les inconvénients :

Pour les systèmes d'information d'entreprise les systèmes client/client restent trop limités. En effet, leur problème majeur est leur manque de séparation entre les différents éléments qui les constituent. C'est également le manque de standards qui a poussé la communauté au concept de «brique» et de séparation des tiers afin d'optimiser leurs développements.

Les limites sont vite apparues :

- Problèmes de performances lorsqu'un grand nombre d'utilisateurs se connecte en parallèle.
- Impossibilité d'écrire des traitements complexes au niveau du serveur.
- Dispersion des ressources.
- Coexistence d'architectures hétérogènes.
- Applications figées, fermées, isolées.
- Mauvaise tolérance aux pannes et peu de mécanismes de répartition de charge : le lien entre client et serveur est direct, ce qui provoque un blocage complet du système lorsque l'un des éléments est défaillant.
- Difficulté de déploiement et installation systématique de l'application et des couches d'accès au SGBD sur chaque poste utilisateur, ce qui implique une taille et une complexité importantes du client.

III. Système distribué :

III.1. Définition 1 :

Un système distribué est une collection d'ordinateurs indépendants qui apparait comme un seul système pour ses utilisateurs.

Un système distribué est généralement séparé en plusieurs composantes entièrement autonomes. Il n'existe pas de composante maitre qui gère les autres et chacune est donc

responsable de son propre fonctionnement. Cela permet, entre autres, d'avoir une hétérogénéité dans la technologie utilisée pour chaque composante, elles peuvent être écrites dans différents langages de programmation (Java, Cobol, C++, etc.) et s'exécutent sur différents systèmes d'exploitation (Mac OS X, Linux, Windows, etc.). L'autonomie des composantes fait que les systèmes sont exécutés simultanément (programmation concurrente). De plus, contrairement au système centralisé, le système distribué possède beaucoup de défaillances (problème de composantes, réseaux, trafics, etc.).[3]

III.2. Définition 2 :

Le modèle n-tiers (3 niveaux) est une extension du modèle Client/serveur dans lequel un processus Serveur peut à son tour devenir le Client d'un autre processus au cours du traitement de la requête. Avec le développement croissant des communications, en particulier du réseau Internet, Ce modèle est aujourd'hui largement répandu.[4]

III.3. Architecture distribuée :

L'architecture d'un environnement informatique ou d'un réseau est dite distribuée quand toutes les ressources ne se trouvent pas au même endroit ou sur la même machine. On parle également d'informatique distribuée. Ce concept s'oppose à celui d'architecture centralisée dont une version est l'architecture client-serveur.

Internet est un exemple de réseau distribué puisqu'il ne possède aucun nœud central. Les architectures distribuées reposent sur la possibilité d'utiliser des objets qui s'exécutent sur des machines réparties sur le réseau et communiquent par messages au travers du réseau. [5]

Dans une architecture distribuée typique les fonctions sont réparties entre un système client (station de travail, terminal,…) et un système serveur (serveur PC, Unix, mainframe**…).**

Chaque système contient une partie de l'application, les parties manquantes sont exécutées sur les autres systèmes participants à l'application et les informations sont échangées par le réseau. [6]

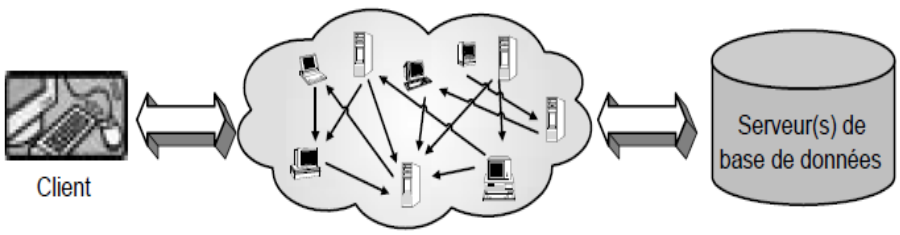

Figure 02 : L'architecture distribuée

Dans cette illustration, vous constaterez la forte indépendance géographique entre les différents éléments qui sont éclatés sur les différents ordinateurs. L'ensemble du dialogue entre ces machines est alors pris en charge par le middleware choisi.

IV. Qu'est-ce que le middleware :

On appelle middleware (ou logiciel médiateur en français) littéralement «élément du milieu» l'ensemble du couches réseau et services logiciel qui permettent le dialogue entre les différents composants d'une application répartie ce dialogue se base sur un protocole applicatif commun défini par l'API de middleware. **[7]**

Le middleware est définit comme une interface de communication universelle être processus

IV.1. Les objectifs principaux du middleware :

- Fourniture interface ou API de haut niveau aux applications.
- Masque l'hétérogénéité des systèmes matériels et logiciels sous-jacents.
- Rendre la répartition aussi invisible (transparente) que possible.
- Facilité la programmation répartie (développement, évolution, réutilisation, portabilité

Des applications)

Sans ce mécanisme la programmation d'une application client-serveur serait complexe et difficilement évolutive.

IV.2. Les services des middlewares :

Les middlewares fournissent les services suivants :

Conversion : service utilisé pour la communication entre machines mettant en œuvre des formats de données différents.

Adressage : permet d'identifier la machine serveur sur laquelle est localisé le service demandé afin d'en déduire le chemin d'accès dans la mesure du possible.

Sécurité : permet de garantir la confidentialité et la sécurité des données à l'aide de mécanismes d'authentification et de cryptage des informations.

Communication : permet la transmission des messages entre les deux systèmes sans altération. Ce service doit gérer la connexion au serveur, la préparation de l'exécution des requêtes, la récupération des résultats et la déconnexion de l'utilisateur. **[8]**

IV.3. Nombreux middlewares :

- CORBA (Common Object Request Broker Architecture).

- DCE (Distributed Computing Environment).
- DCOM (Distributed Component Object Model).
- RMI (Remote Method Invocation).
- .NET Remotin

V. Les différentes architectures (serveur/client) :

V.1. L'architecture 3 tiers :

Cette architecture trois tiers (3 niveaux), également appelée client-serveur de deuxième génération ou client-serveur distribuée séparé. Elle place un intermédiaire entre le client et le serveur, dit serveur de traitement ou serveur applicatif, qui exécute directement les traitements et transmet les requêtes au serveur de base de données.

L'application en 3 niveaux de services distincts, conformes au principe suivant :

- ***Client :*** couche présentation (interface homme/machine).
- ***Serveur d'application :*** ensemble des traitements applicatifs.
- ***Serveur de bases de données :*** stockage des données.

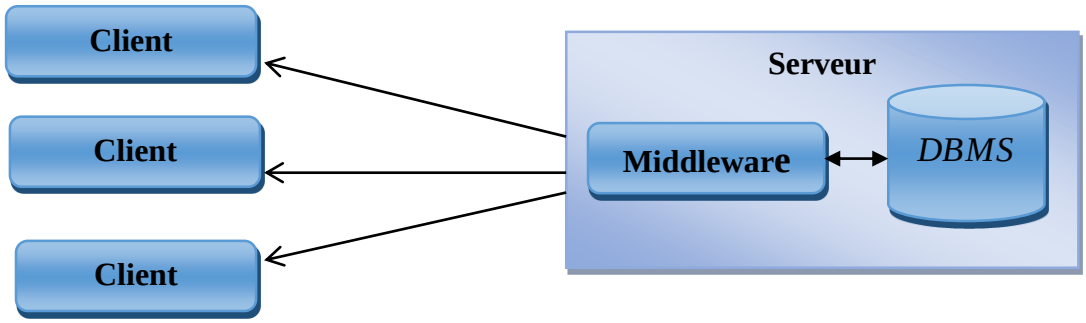

Figure 03 : L'architecture 3 tiers.

V.1.1. Intérêt potentiel :

- séparation des fonctions.
- interfaces bien définies, normalisation, ouverture.
- capacité d'évolution et de croissance.
- réutilisation de l'existant.

V.2. L'architecture n-tiers :

L'architecture n-tiers a été pensée pour pallier aux limitations des architectures trois permet de distribuer plus librement la logique applicative, ce qui facilite la répartition de la charge entre tous les niveaux.

Cette évolution des architectures trois tiers met en œuvre une approche objet pour offrir une théoriquement, ce type d'architecture supprime tous les inconvénients de l'architecture précédente, elle :

- permet l'utilisation d'interfaces utilisateurs riches.
- sépare nettement tous les niveaux de l'application.
- offre de grandes capacités d'extension.
- facilite la gestion des sessions.

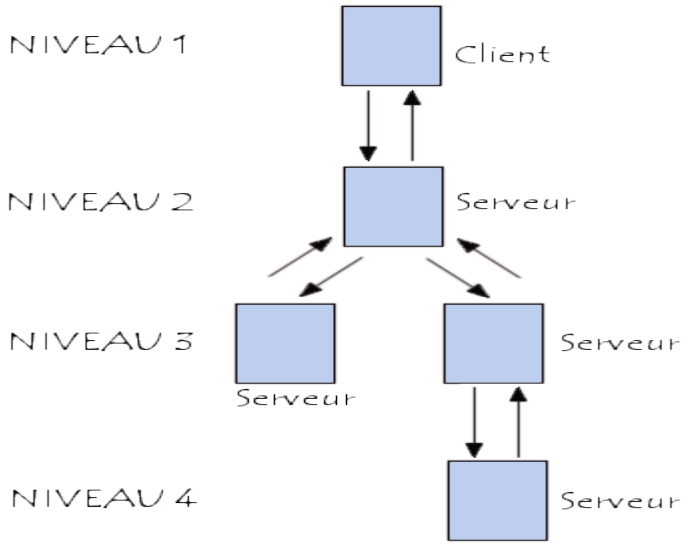

Figure 04 : L'architecture n-tiers

VI. Les composants de système distribué :

- Obtenir quelques informations de la machine A et les amener sur la machine B, ou vice versa. Ceci est réalisé au moyen de la programmation réseau de base.
- se connecter à une base de données, qui peut résider quelque part sur le réseau. Pour Cela, on utilise JDBC, qui est une encapsulation des détails confus, et spécifiques à la plate-forme, du SQL.
- Fournir des services via un serveur Web. C'est le rôle des servlets Java et des JSP.
- Exécuter de manière transparente des méthodes appartenant à des objets Java résidant Sur des machines distantes, exactement comme si ces objets résidaient sur la machine locale. Pour cela, on utilise le RMI.

- Utiliser du code écrit dans d'autres langages, tournant sous d'autres architectures. C'est l'objet de CORBA, qui est directement mis en œuvre par Java.
- Séparer les questions relatives à la connectivité de la logique concernant le résultat Recherché, et en particulier les connexions aux bases de données incluant la gestion des transactions et la sécurité. C'est le domaine des EJB. Les EJB ne représentent pas réellement une architecture distribuée, mais les applications qui en découlent sont couramment utilisées dans un système client-serveur en réseau.
- Ajouter et enlever, facilement et dynamiquement, des fonctionnalités provenant d'un réseau considéré comme un système local. C'est ce que propose la fonctionnalité Jini de Java.**[9]**

VII. Les objectives du système distribue :

Les systèmes distribués sont facilement utilisables pour plusieurs raisons :

- ***Accès distant :*** un même service peut être utilisé par plusieurs acteurs, situés à des endroits différents
- ***Redondance :*** des systèmes redondants permettent de pallier une faute matérielle, ou de choisir le service équivalent avec le temps de réponse le plus court
- ***Performance :*** la mise en commun de plusieurs unités de calcul permet d'effectuer des calculs parallélisables en des temps plus courts.
- **Confidentialité :** les données brutes ne sont pas disponibles partout au même moment, seules Certaines vues sont exportées.
- ***Coût :*** plusieurs processeurs à bas prix.
- ***Puissance de calcul et de stockage :*** aucune machine centralisée ne peut la rivaliser**.**
- ***Adaptation :*** à des classes d'applications réelles naturellement distribuées.
- ***Fiabilité :*** résistance aux pannes logicielles ou matérielles**.**
- ***Extensibilité :*** croissance progressive selon le besoin. **[10]**

VIII. Domaines et exemples d'application :

Le « système distribue » compte plusieurs domaines d'utilisation. Il est important de noter que les entreprises, notamment les plus grandes, possèdent déjà généralement un environnement informatique fonctionnel. Les applications distribuées s'intègrent donc en toute transparence dans les systèmes existants. Nous décrirons par la suite des cas d'utilisation possible.

- Peer to Peer (bit torrent).
- Web (http).

VIII.1. Le Peer-to-Peer (poste à poste) :

Le modèle Peer-to-Peer est un exemple de réussite des architectures distribuées où chaque ordinateur est à la fois serveur de données et client des autres. Ce modèle peut être appliqué au partage de ressource. Son principe est de mettre directement en liaison un internaute avec un autre internaute qui possède un fichier convoité. Il existe 2 méthodes pour accomplir cette tâche :

- ***La méthode centralisée :*** est basée sur un ou plusieurs serveurs qui possèdent la liste des fichiers partagés et qui orientent les internautes vers l'internaute possédant le fichier convoité.
- ***La méthode décentralisée :*** utilise chaque internaute comme un mini-serveur et ne possède aucun serveur fixe. Cette méthode a l'avantage de répartir les responsabilités et d'éviter les actions en justice**.**[11]

VIII.2. Web (http) :

(Hyper Text Transfert Protocol) Protocole sans état, rapide, léger, de niveau d'application permettant le transfert de données d'une connexion de type TCP (sure).

- Utilise les URL.
- Port par défaut 80.
- Basé sur le paradigme de requête / réponse.
- Chaque requête et chaque réponse est un « message HTTP ».
- Largement utilisé sur le World Wide Web.[12]

IX. Intérêt fonctionnels :

- Décentralisation des responsabilités.
- Optimisation de l'utilisation des ressources (répartition de charge).
- Amélioration des performances.
- Fiabilité (redondance).
- Flexibilité.

Conclusion :

Dans ce chapitre nous avons défini ce qu'est le système distribue, ses types selon son mode de déploiement ou bien selon les services délivrés, puis Les avantages et les inconvénients, en suite son domaine d'application.

Dans le chapitre qui suit nous allons présenter les technologies de la programmation distribuée.

Chapitre 02 : Les technologies de programmation distribuée

Introduction :

Ce chapitre permet d'introduire les technologies de programmation distribuée : RMI de Sun microsystème, COBRA de l'objet Management Group, le Framework .NET de Microsoft en fin la représentation des composants EJB (entreprise Java Bean) les plus important de la plateforme Java EE pour le développement d'application distribuée.

I. RMI :

I.1. Définition :

RMI est une API Java permettant de manipuler des objets distants de manière transparente pour l'utilisateur, c'est-à-dire de la même façon que si l'objet était sur la machine virtuelle (JVM) de la machine locale. Facilite la mise en œuvre et l'utilisation d'objets distants java, et préserve la sécurité (RMI Security Manager, DGC). Ainsi un serveur permet à un client d'invoquer des méthodes à distance sur un objet qu'il instancie. Deux machines virtuelles sont donc nécessaires une sur le serveur et une sur le client et l'ensemble des communications se fait en Java. [13]

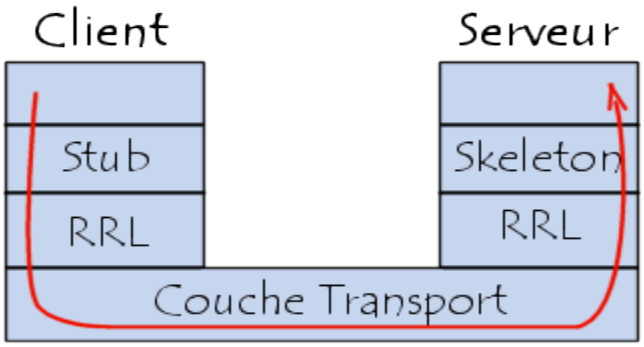

Figure 05 : Processus de fonctionnement de RMI

I.2. Architecture de RMI :

L'architecture RMI définit la manière dont se comportent les objets, comment et quand des exceptions peuvent se produit, comment gérer la mémoire et comment les méthodes appelées passent et reçoivent les paramètres.

Le système RMI contient 3 couches qui sont :

- La couche des amorces (stub/Skelton).
- La couche des références (RRL).
- La couche de transport.

Chacun de ses couches est indépendante de l'autre et utilise un protocole spécifique, la transmission des objets utilise deux technique : **[14]**

- La sérialisation.
- Le chargement dynamique du stub, permet au client de charger dynamiquement

Le stub quand il a seulement l'interface.

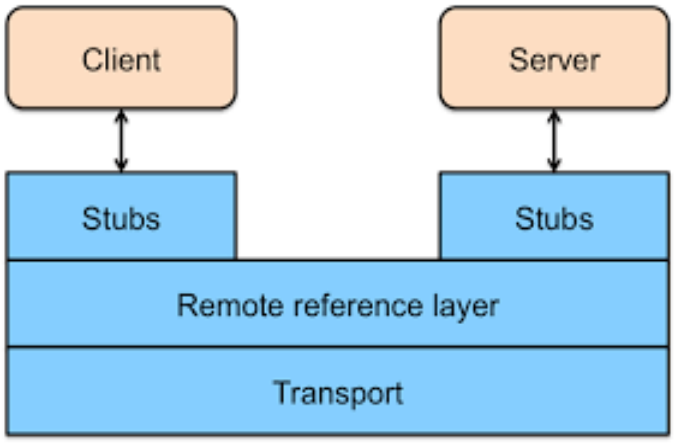

Figure 06 : Architecture de RMI

-1ère couche : Stub/Skelton

Le stub (traduisez souche) et le Skelton (traduisez squelette), respectivement sur le client et le serveur, assurent la conversion des communications avec l'objet distant.

- ***Stub (coté client) :*** est un mandataire de l'objet qui est chargé dans le client au moment de l'obtention de la référence, cette référence implémente les mêmes interfaces que l'objet distante, le stub a pour but de :
- Initie une connexion avec la JVM distante en transmettant l'invocation distante à la couche des références d'objets (RRL).
- Assemble les paramètres pour leur transfert à la JVM distante.
- Attend les résultats de l'invocation distante, désassemblent la valeur ou l'exception renvoyée, et renvoient la valeur à l'appelant.
- ***Skelton (coté serveur) :*** contient une méthode qui appelle les méthodes de l'objet distant.

Il a pour but de :

- Désassemblent les paramètres pour la méthode distante.
- Font l'appel à la méthode demandée.
- Assemblage du résultat (valeur renvoyée ou exception) à destination de l'appelant.

-2ème couche : RRL (couche des références d'objets)

La couche de référence (RRL, Remote Reference Layer) est chargée du système de localisation afin de fournir un moyen aux objets d'obtenir une référence à l'objet distant.

Elle est assurée par le package java.rmi.Naming. On l'appelle généralement registre RMI car elle référence les objets.

Rmiregistry s'exécute sur chaque machine hébergeant des objets distants.

-3ème couche : Transport

La couche de transport permet d'écouter les appels entrants ainsi que d'établir les connexions et le transport des données sur le réseau par l'intermédiaire du protocole TCP. Les packages java.net.Socket et java.net.SocketServer assurent implicitement cette fonction. Les connexions et les transferts de données dans RMI sont effectués par Java sur TCP/IP grâce à un protocole propriétaire JRMP, (Java Remote Method Protocol) sur le port 1099.

A partir de Java 2 version 1.3, les communications entre client et serveur s'effectuent grâce au protocole RMI-IIOP (Internet Inter-Orb Protocol), un protocole normalisé par l'OMG et utilisé dans l'architecture CORBA.

Généralement lors d'un appel d'une méthode, le Stub transmet l'appel à la couche RRL qui assure la connexion avec le serveur en point à point (unicast).

Cette couche transmet la requête à la couche transport qui utilise JRMP au-dessus de TCP/IP et transfère la requête en remontant vers le skeleton qui appelle la méthode sur l'objet distant.

Avec Java 2, le skeleton est devenu obsolète (dépassé), une même classe skeleton générique est partagée par tous les objets distants. En plus, jusqu'à la version 5.0 du J2SE (2005), il fallait utiliser un compilateur de stub appelé RMIC (Java RMI Compiler) pour générer les stub/skeleton avant tout enregistrement sur le registre RMI. **[14]**

I.3. Avantages et inconvénients de RMI :

Parmi les avantages de RMI :

- Il y a une apparence superficielle de la simplicité.
- la distance peut être traitée comme si elle était locale.
- la simplicité de programmation.
- son utilisation est sécuritaire.
- il élimine le coulage de mémoire. **[15]**

Les inconvénients importants de RMI sont :

- Utilisation exclusive avec JAVA :

Pas d'interopérabilité avec d'autres langages de programmation.

- Relative lenteur à l'exécution :

Due aux coûts de la sérialisation des paramètres.

- défaillance du réseau.
- Architecture fortement couplé. [15]

II. CORBA :

II.1. Définition :

CORBA (Common Object Request Broker Architecture) est une architecture proposée et maintenue par l'OMG (Object Management Group) pour permettre l'intégration d'une grande variété de systèmes hétérogènes distribués orientés objet. CORBA est la spécification d'architecture la plus répandue pour développer des systèmes distribués.

La première version de CORBA (version 1.1) date de 1992. CORBA a toujours été en pleine évolution pour prendre en compte de plus en plus de besoins d'utilisateurs. La version actuelle, CORBA 3.0, a été adoptée fin 2002 et éditée en 2004.

CORBA est conçue pour supporter des applications distribuées orientées objet et développées selon le modèle client-serveur. On dit aussi que CORBA est un bus
Réparti (à ne pas confondre évidemment avec la notion de bus physique utilisée dans les réseaux de transmission). [16]

Le bus CORBA propose un modèle orienté objet client/serveur d'abstraction et de de-coopération entre les applications réparties :

- La composante d'abstraction : Chaque application peut exporter certaines de ses services sous la forme d'objets CORBA
- La partie coopération : Les interactions entre les applications sont alors matérialisées par des invocations à distance des méthodes des objets. [17]

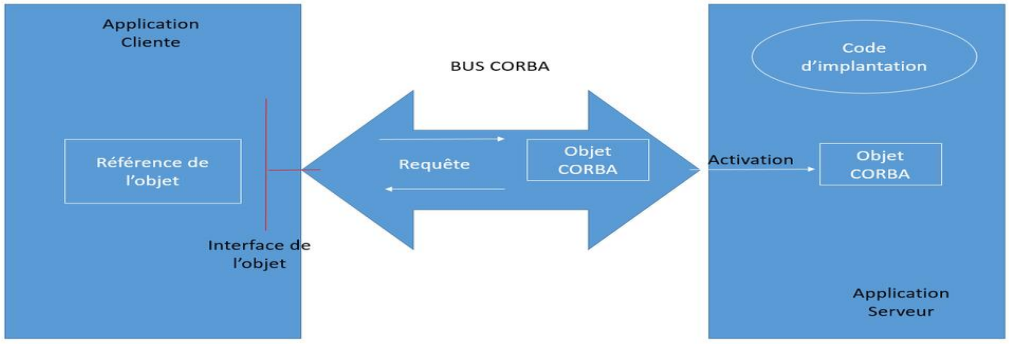

Figure 07 : Notion client/serveur avec le bus de CORBA

Ce modèle d'objet intervient uniquement lors de l'utilisation d'un objet. On caractérise alors l'application implantant l'objet comme le serveur et l'application utilisant l'objet comme le client. Cependant il faut noter qu'une application peut être à la fois cliente et serveur.
Sur le schéma précédent on voit différentes notions :

- L'application cliente : Programme qui utilise les méthodes objets via le bus CORBA
- La référence de l'objet : Structure qui désigne l'objet CORBA et permet de le localiser sur le bus.
- L'interface de l'objet : Ce qui définit les opérations et attributs de l'objet CORBA (utilisation du langage OMG-IDL)
- Le bus CORBA : Transfert les requêtes de l'application cliente vers l'objet
- L'objet CORBA : Entité virtuelle gérée par le bus CORBA.
- L'activation : Technique d'association d'un objet d'implantation à un objet CORBA
- Le code d'implantation : Regroupement des traitements liés à l'implantation de l'objet CORBA
- L'application serveur : Structure d'accueil des objets d'implantation et des exécutions des opérations. **[17]**

II.2. Principe de CORBA :

- Bus logiciel ORB.
- Oriente objet (Héritage, Polymorphisme, Encapsulation)
- Basé sur une entité virtuelle : l'objet CORBA.
- Déroulement de la coopération client-serveur.
- Une séparation stricte Interface/Implémentation.
- La transparence de la localisation des objets.
- La transparence de l'accès aux objets.
- Le typage des Object Références par les interfaces.
- L'héritage multiple d'interfaces

II.3. Architecture CORBA :

Cette architecture, assure l'automatisation des tâches de communication, de localisation et d'activation d'objets, et la traduction des messages échangés entre systèmes hétérogènes.

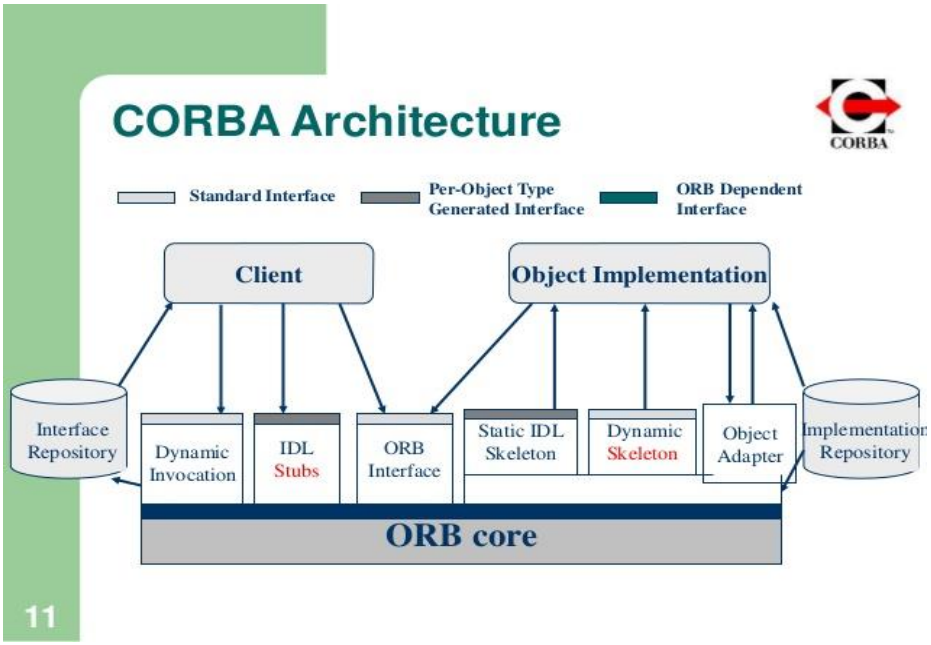

Figure 08 : Architecture CORBA

Etant une spécification, CORBA n'est lié à aucune plate-forme matérielle ou logicielle, ni à aucun langage de programmation particulier. On peut donc en trouver des implémentations de tous horizons, telles qu'Open ORB, Mico, Orbacus et JacORB. CORBA est associé un langage de définition d'interface (IDL). Celui-ci permet de décrire les services fournis par un objet, de manière indépendante du langage de programmation utilisé pour son implémentation.

Un compilateur d'IDL permet alors de traduire une interface IDL vers un langage de programmation particulier. Cette étape va générer des portions de code parmi lesquelles le stub et le skeleton, qui assurent de liaison entre l'environnement d'exécution, l'implémentation proprement dite de l'objet serveur et le client.

Dans le monde CORBA, un objet serveur est vu comme une instance d'interface IDL et sa localisation est masquée par un mécanisme de référence qui lui associe un ou plusieurs chemins d'accès par lequel un client peut le contacter. **[18]**

II.4. Les Avantages et les Inconvénients de CORBA :

CORBA présente des avantages importants pour les systèmes distribués comme :

- La transparence.
- La portabilité.
- L'interopérabilité.

- L'adaptabilité.
- La disponibilité.
- La stabilité.

Les inconvénients de CORBA sont représentés par :

- La complexité.
- Le prix élevé
- Une formation spécialisée pour les développeurs. **[19]**

II.5.Objectifs :

Le standard CORBA offre une solution ouverte et évolutive avec une architecture modulaire garantissant l'interopérabilité entre des composant hétérogènes, tout en gardant un choix libre pour les technologies d'implantation.

III. .NET :

III.1. Définition :

Microsoft .NET est un terme général qui désigne un certain nombre de technologies lancées par Microsoft. Prises dans leur ensemble, ces technologies représentent les changements les plus importants apportées à la plate-forme de développement Microsoft depuis le passage du développement 16 bits au développement 32bits. La plate-forme Microsoft .NET est une solution complète pour développer, déployer et exécuter des Applications de tous types, y compris des Services Web. Fondée sur des standards de l'industrie (HTTP, XML, SOAP, WSDL). La plate-forme .NET est un moyen simple et puissant d'implémenter la coopération des services logiciels entre eux, quelle que soit leur localisation, leur implémentation technique, qu'ils soient internes ou externes, existants ou à inventer.

Comme elle repose sur le système d'exploitation Windows et composée du Framework .NET qui est le socle de développement.**[20]**

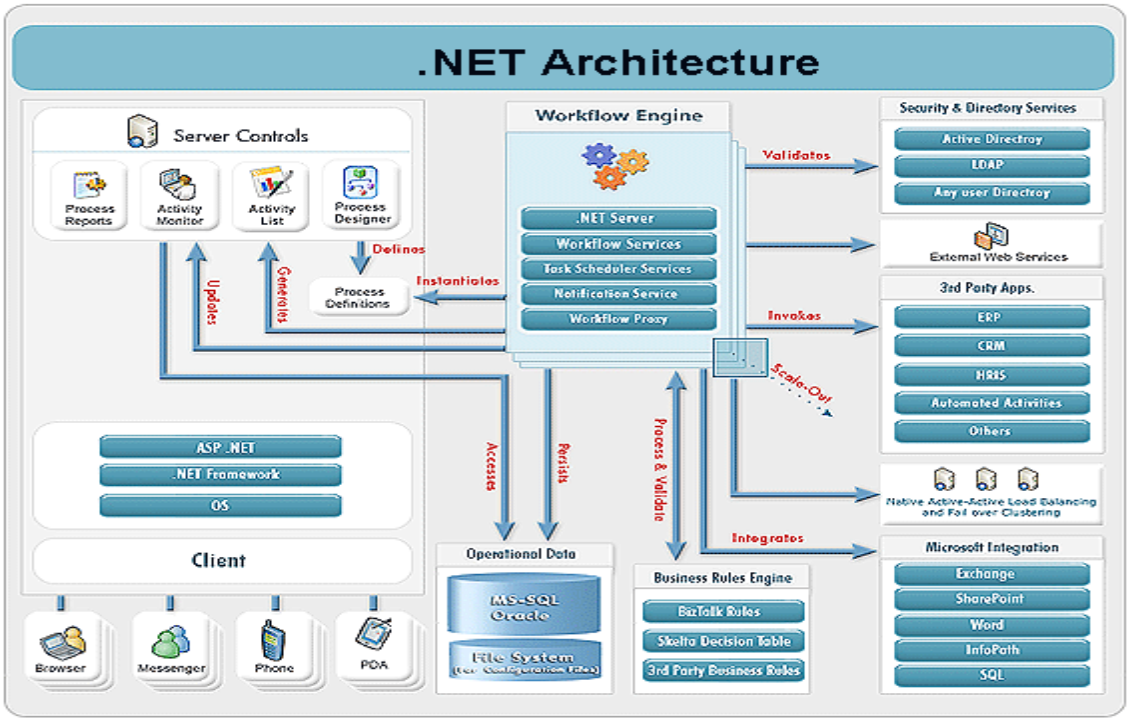

Figure 09 : L'architecture de .NET

III.2. Le Framework .NET :

Le Framework .NET est un ensemble de technologies servant de base pour développement d'applications. Il se compose de trois grandes parties :

- Le CLR ou Runtime est l'équivalent de la JVM en Java. Avec les langages .NET vous ne compilez plus le code des programmes directement en code natif mais dans un langage intermédiaire nommé MSIL qui est l'équivalent du byte-code en Java. Ce code est ensuite pris en charge par le Runtime : celui-ci se charge de la compilation finale au moment de l'installation ou de la première exécution du programme, gère la mémoire (Garbage collector) et la durée de vie des objets.

Le .NET Framework n'étant pas sensible à un langage particulier, y compris le langage c# introduit avec .NET.

- Les bibliothèques de classes représentent une manne de fonctionnalités dans lesquelles les programmeurs des langages .NET peuvent puiser.
- ASP.NET, la nouvelle version d'ASP. **[21]**

Figure 10 : L'architecture de .NET Framework

Le Framework gère tous les aspects de l'exécution d'une application dans un environnement d'exécution dit « manager » :

- il alloue la mémoire pour le stockage des données et des instructions du programme.
- il autorise ou refuse des droits à l'application.
- il démarre et gère l'exécution.
- il gère la réallocation de la mémoire pour les ressources qui ne sont plus utilisées.

Il est composé de deux blocs principaux :

- la mise en œuvre d'une machine virtuelle compatible CLI sous le nom CLR et DLR.
- le Framework .NET.

III.3. Objectifs de .NET :

L'objectif de .NET est d'assurer l'interopérabilité entre les différentes machines reliées sur internet. Il s'agit de pouvoir rassembler l'information, et de profiter des services de l'internet sur plusieurs types de supports. .NET veut être la plate-forme de référence pour le développement de la prochaine génération d'application : les services Web XML. Fournir un serveur web local permettant de gérer des services et évitant d'externaliser des données privées sur un service web de stockage ou un hébergement web tiers. **[21]**

III.4. Principales caractéristiques de .NET :

III.4.1. Interopérabilité :

Du fait de la nécessité de pouvoir interagir avec les anciennes applications, le Framework fournit des moyens pour accéder aux fonctionnalités en dehors de l'environnement .NET. **[22]**

III.4.2. Common Runtime Engine :

Les langages de programmation du Framework sont compilés dans un langage intermédiaire appelé CIL. Ce langage n'est pas interprété, mais subit une compilation à la volée et une compilation au niveau de la CLR. La CLR est l'implémentation de la CLI. **[22]**

III.4.3. Indépendance du langage :

La spécification du CTS définit l'ensemble des types de données et structures de programmation supportés par la CLR ainsi que leurs interactions. Par conséquent, le .NET Framework supporte l'échange des instances des types entre les programmes écrits dans un des langages .NET. **[22]**

III.5. Structure d'une application .NET :

L'unité de base d'une application .NET est appelée un assemblage (assembly). Il s'agit d'un ensemble de code, de ressources et de métadonnées. Un assemblage est toujours accompagné par un manifeste (assembly manifest) qui décrit ce qu'il contient : nom, version, types de données exposées, autres assemblages utilisés, instructions de sécurité. Un assemblage est composé d'un ou plusieurs modules qui contiennent le code. **[23]**

III.6. Avantage :

- .NET va être exécuté plus rapidement qu'un langage interprété puisqu'il aura été au préalable précompilé en un langage déjà proche de la machine.
- CLR permet au développeur de ne pas se préoccuper de certaines choses, surtout concernant la mémoire. Un « garbage collector » va s'occuper de libérer les ressources mémoires inutilisées, les fuites de mémoires vont être automatiquement détectées et corrigées.
- Peu importe le langage (C #, VB.NET, F#, etc…) que l'on utilise, le langage intermédiaire sera exactement le même.
- Peu importe la plateforme Windows que vous avez (XP, Vista, Windows server, etc.⋯) il vous suffit d' avoir la CLR pour que votre programme fonctionne.**[24]**

IV. J2EE :

IV. 1. Définition :

J2EE c'est une norme proposée par la société Sun, portée par un consortium de sociétés internationales, visant à définir un standard de développement d'applications d'entreprises basé sur le langage Java. Chaque vendeur pourra offrir une plateforme

correspondant à ce standard. La plate-forme J2EE est un ensemble constitué des services (API) offerts et de l'infrastructure d'exécution. Comprend notamment les spécifications du serveur d'application et des services, au travers d'API. **[25]**

IV.2. Architecture :

L'architecture J2EE permet de séparer la couche présentation, correspondant à IHM, la couche métier contenant l'essentiel des traitements de données en se basant dans la mesure du possible sur des API existantes, et enfin la couche de données correspondant aux informations de l'entreprise stockées dans des fichiers, dans des bases de données relationnelles ou XML, dans des annuaires d'entreprise ou encore dans des systèmes d'information complexes.**[26]**

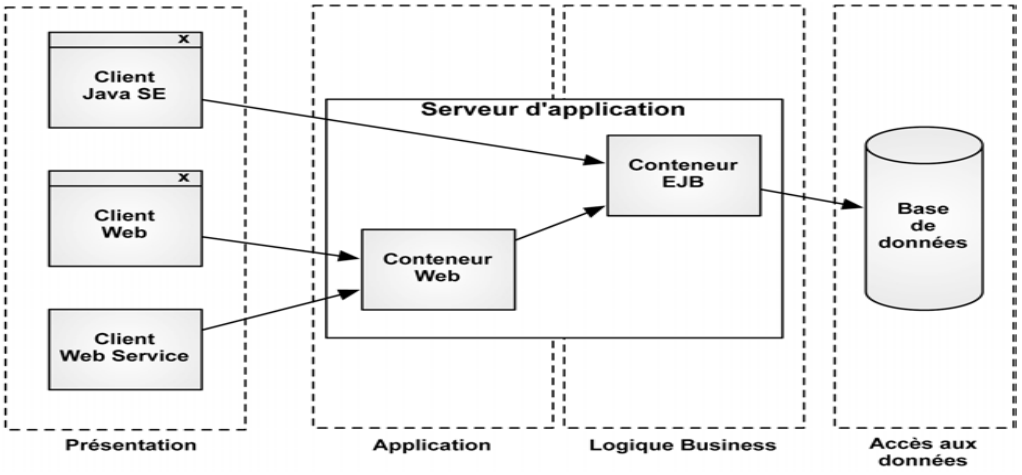

Figure 11 : L'architecture J2EE

La technologie EJB fait partie d'une plate-forme plus large appelée Java EE, Cette plate-forme constitue une architecture pour le développement, le déploiement et l'exécution des applications distribuées.

Ces applications requièrent des services techniques comme la gestion de transactions, la gestion de la sécurité, l'accès par les clients, l'accès aux bases de données, La plate-forme Java EE fournit ces services techniques.

Le développeur peut se concentrer sur le logique métier au lieu de se disperser sur des problèmes techniques, L'architecture logicielle d'une application est, dans de nombreux cas, constituée des couches suivantes : **[27]**

IV.2.1. Couche présentation (client) :

La couche présentation est liée au type de clients utilisés. Si vous souhaitez travailler avec un client riche, vous devrez utiliser l'ensemble des outils et des librairies mis à disposition pour ce type de client (plus particulièrement J2SE).

IV.2.2.Couche application :

La couche application sert de médiateur entre la couche présentation et la couche métier, et contrôle l'enchaînement des tâches. Elle est chargée de connaître et degérer l'état (connecté, en attente, déconnecté…) des sessions des clients connectés, si l'état est de type «Conversationnel ». Cette couche représente le contrôleur dans un modèle MVC.

IV.2.3. Couche métier, ou logique business :

La couche métier est la couche principale de toute application. Elle doit s'occuper aussi bien des accès aux différentes données qu'à leurs traitements, suivant les processus définis par l'entreprise. On parle généralement de traitement métier. Cette expression regroupe :

- la vérification de la cohésion entre les données.
- l'implémentation du logique métier de l'entreprise au niveau de l'application.
- la gestion du workflow.

Dans le domaine de l'assurance, par exemple, un traitement métier pourrait s'apparenter à une méthode de calcul d'une prime d'assurance.

Il est cependant plus propre de séparer toute la partie accès aux données de la partie traitement du logique métier. Cela offre plusieurs avantages. Tout d'abord, les développeurs ne se perdent pas entre le code métier, qui peut parfois être complexe, et le code d'accès aux données, plutôt élémentaire mais conséquent. Cela permet aussi d'ajouter un niveau d'abstraction sur l'accès aux données et donc d'être plus modulable en cas de changements de type de stockage.

Il est alors beaucoup plus facile de se répartir les différentes parties au sein d'une équipe de développement. Dans le cas d'applications Java EE de grande envergure, cette couche est représentée par les EJB (Entreprise JavaBeans).

Ces composants sont particulièrement adaptés aux panels hétérogènes de clients (Web, client riche, client d'un autre langage…) devant se connecter à une même logique métier. Si votre application n'a, cependant, besoin que d'un seul type de client, il vous sera alors

possible d'étudier d'autres solutions moins «gourmandes» en ressources. Hibernate, Spring, iBatis pour n'en citer que quelques-unes.

IV.3. Les composants JEE :

IV.3.1. JPA :

La persistance des objets en Java a longtemps donné lieu à des manipulations fastidieuses. Aujourd'hui, la *Java Persistance API* apporte une solution performante et simple à utiliser. S'appuyant sur JDBC (*Java Data Base Connectivity*), JPA propose une abstraction suffisante pour que le développeur n'ait, dans la majorité des cas, pas à se préoccuper du fonctionnement de JDBC.

IV.3.2 Servlet :

Les Servlets forment l'un des composants JEE les plus utilisés. Elles permettent de gérer des requêtes HTTP et de fournir au client une réponse HTTP et forment ainsi la base de la programmation Web JEE.

Les Servlets s'exécutent toujours dans un moteur de Servlet ou conteneur de Servlet permettant d'établir le lien entre la Servlet et le serveur Web. **[28]**

IV.3.3. Pages JSP :

Les JSP (*Java Server Pages*) sont des pages HTML qui permettent l'affichage de contenu web dynamique par l'intégration de fragments de code Java et de directives JSP. **[28]**

IV.3.4. JSF :

Les JSF (Java Server Faces) est un Framework Java basé sur la notion de composants où l'état d'un composant est enregistré lors du rendu de la page, pour être ensuite restauré au retour de la requête. Il utilise JSP par défaut, mais peut être utilisé avec d'autres technologies.**[28]**

IV.3.5. EJB 3.0 :

Les EJB (*Enterprise Java Beans*) forment l'une des spécifications majeures de JEE et utilisent le principe des annotations Java. Ils se déclinent en :

- entités (*Entity Bean*).
- sessions (*Session Bean*).
- messages (*Message Bean*). **[28]**

V. EJB :

V.1. Définition :

Les EJB sont des composants serveurs donc non visuels qui respectent les spécifications d'un modèle édité par Sun. Ces spécifications définissent une architecture, un environnement d'exécution et un ensemble d'API. Le respect de ces spécifications permet d'utiliser les EJB de façon indépendante du serveur d'applications J2EE dans lequel ils s'exécutent, du moment où le code de mise en œuvre des EJB n'utilise pas d'extensions proposées par un serveur d'applications particulier. Le but des EJB est de faciliter la création d'applications distribuées pour les entreprises. Une des principales caractéristiques des EJB est de permettre aux développeurs de se concentrer sur les traitements orientés métiers car les EJB et l'environnement dans lequel ils s'exécutent prennent en charge un certain nombre de traitements tel que la gestion des transactions, la persistance des données, la sécurité, ... Physiquement, un EJB est un ensemble d'au moins deux interfaces et une classe regroupées dans un module contenant un descripteur de déploiement particulier. **[29]**

Il existe plusieurs versions des spécifications des EJB : De la version 1.0 à la version 2.1, un EJB était accompagné d'un ou plusieurs fichiers de déploiement écrit en XML qui permettait au serveur applicatif de déployer correctement l'objet au sein d'un conteneur. C'était notamment dans ces fichiers de déploiement que le développeur avait la possibilité de préciser le cadre transactionnel dans lequel l'objet allait s'exécuter. Depuis la version 3.0, le modèle EJB utilise le principe d'annotation java (métadonnées) pour spécifier toute la configuration et les propriétés transactionnelles de l'objet. Le fichier de code source de l'EJB se suffit à lui-même. **[30]**

EJB est une architecture de composants logiciels côté serveur pour la plateforme de développement JEE. Cette architecture propose un cadre pour créer des composants distribués (déployés sur des serveurs distants) écrit en langage de programmation java hébergés au sein d'un serveur applicatif permettant de représenter des données (EJB dit entité), de proposer des services avec ou sans conservation d'état entre les appels (EJB dit session), ou encore d'accomplir des tâches de manière asynchrone (EJB dit message).

V.2. L'architecture EJB :

Une architecture EJB est composée d'au moins 3-tiers : une machine cliente, une machine contenant la logique applicative et une base de données.

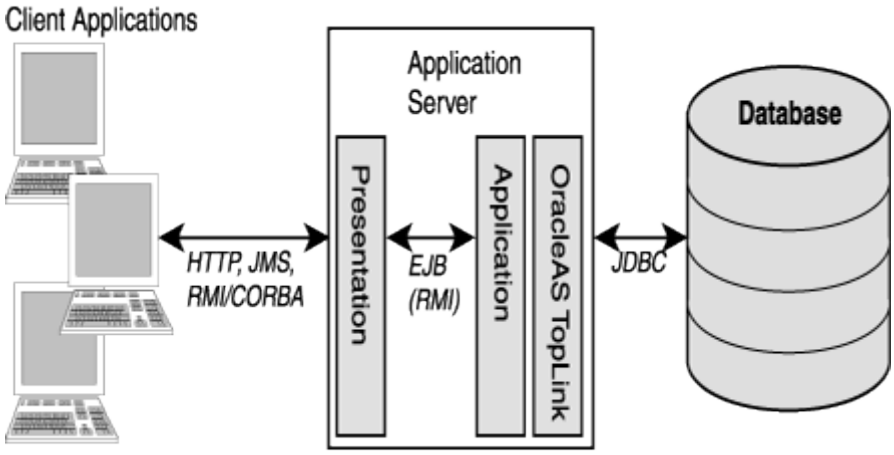

Figure 12 : Architecture EJB simple

Dans ce type d'architecture, les EJB se situent dans le tiers « logique applicative ». Les EJB implémentent le logique métier de l'application et représentent une abstraction de l'accès à la base de données. Cependant, l'architecture EJB est avant tout une architecture évolutive. Il est alors possible de faire évoluer une application EJB simple vers une application orientée Business To Business connectant vers d'autres systèmes d'entreprise.

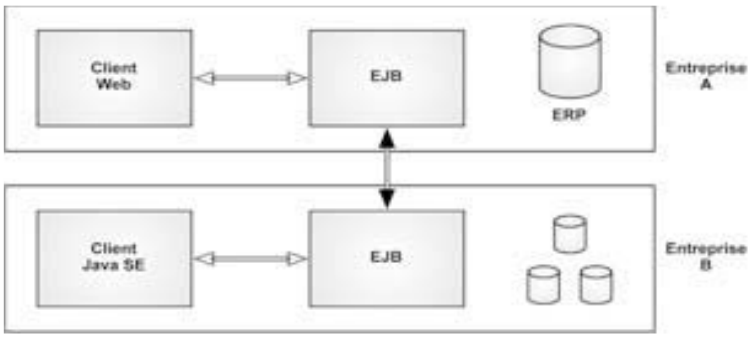

Figure 13: Architecture EJB type business to business

Les EJB sont des applications exécutées côté serveur mais également englobées dans un conteneur. Chaque partie a ses propres obligations et règles, ce qui permet de faire évoluer une partie du serveur sans avoir à tout faire évoluer.

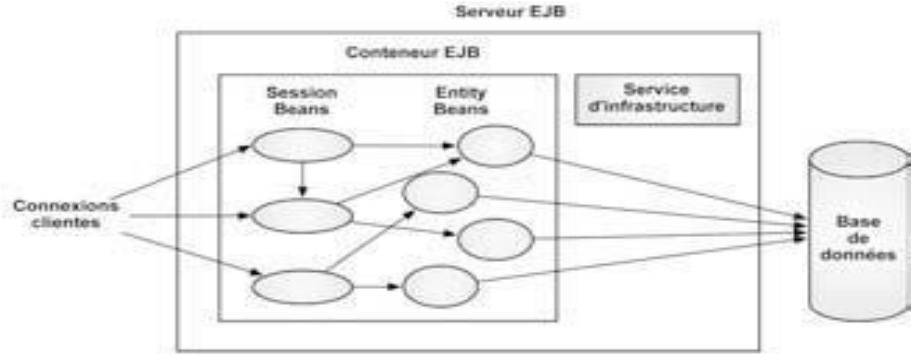

Figure 14 : Serveur EJB

La figure présente la structure d'un serveur EJB. Même si le conteneur a le rôle le plus important, c'est le serveur qui aiguille l'ensemble des requêtes et gère l'ensemble des conteneurs et services. Le serveur se doit de gérer, de plus, un ensemble de services d'infrastructure communs à l'ensemble des conteneurs ou des services.

La spécification Java EE oblige le serveur à offrir un service d'annuaire JNDI et un service de transaction. Bien entendu, les serveurs d'applications fournissent généralement d'autres services facilitant le développement des applications.

De façon imagée, on peut considérer le serveur EJB comme un orchestre. Le rôle de ce serveur est celui du chef d'orchestre. C'est lui qui dirige l'ensemble des services et leur cycle de vie (démarrage, arrêt, pause...). Chaque partie de l'orchestre correspond à un service (EJB, transaction, base de données...). Ils sont tous indépendants, mais ils travaillent ensemble pour produire un résultat commun. **[31]**

V.3. La présentation des EJB :

Les EJB sont des composants et en tant que tels, ils possèdent certaines caractéristiques comme la réutilisabilité, la possibilité de s'assembler pour construire une application, etc. ...

Les EJB et les beans n'ont en commun que d'être des composants. Les JavaBeans sont des composants qui peuvent être utilisés dans toutes les circonstances. Les EJB doivent obligatoirement s'exécuter dans un environnement serveur dédié.

Les EJB sont parfaitement adaptés pour être intégrés dans une architecture trois tiers ou plus. Dans une telle architecture, chaque tier assure une fonction particulière :

- le client « léger » assure la saisie et l'affichage des données.
- sur le serveur, les objets métiers contiennent les traitements. Les EJB sont spécialement conçus pour constituer de telles entités.
- une base de données assure la persistance des informations.

Les EJB s'exécutent dans un environnement particulier : le serveur d'EJB. Celui-ci fournit un ensemble de fonctionnalités utilisées par un ou plusieurs conteneurs d'EJB qui constituent le serveur d'EJB. En réalité, c'est dans un conteneur que s'exécute un EJB et il lui est impossible de s'exécuter en dehors. **[32]**

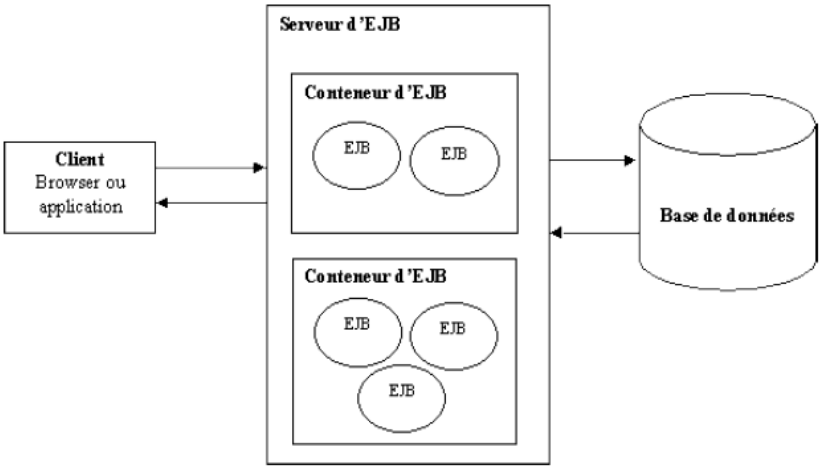

Figure 15 : L'exécution d'EJB

Le conteneur d'EJB propose un certain nombre de services qui assurent la gestion :

- du cycle de vie du bean.
- de l'accès au bean.
- de la sécurité d'accès.
- des accès concurrents.
- des transactions.

Les entités externes au serveur qui appellent un EJB ne communiquent pas directement avec celui-ci. Les accès à l'EJB par un client se font obligatoirement via le conteneur. Un objet héritant de la classe EJB Object assure le dialogue entre ces entités et les EJB via le conteneur. L'avantage de passer par le conteneur est que celui-ci peut utiliser les services qu'il propose et libérer ainsi le développeur de cette charge de travail. Ceci permet au développeur de se concentrer sur les traitements métiers proposés par le bean.

Il existe de nombreux serveurs d'EJB commerciaux : BEA Weblogic, IBM Webpshere, Sun IPlanet, Macromedia JRun, Borland AppServer, etc. ... Il existe aussi des serveurs d'EJB open source dont les plus avancés sont JBoss et Jonas. **[32]**

V.4. Caractéristiques d'EJB :

- EJB s'intéresse aux activités liées au développement, au déploiement et à l'exécution d'une application.
- EJB définit différents rôles associés aux différentes parties intervenant dans la production d'une application.
- EJB définit des contrats associés à un Bean. **[33]**

V.5. Les principes :

- Les EJBs sont des objets dérivés de la classe de base EJBObject, contenue dans le package javax.ejb. Les EJBs sont définies dans les spécifications EJB 2.0.
- Ce sont des composants capables d'être exécutés en environnement distribué, avec des capacités transactionnelles et une gestion de la persistance.
- La philosophie WORA : "Write Once, RunAnywhere" a guidé la conception des EJBs. Ces composants permettent d'étendre dynamiquement le comportement d'un serveur et de connecter facilement des applications à une BD.
- Utiliser des EJB revient à "enficher" des entités autonomes sur une plate-forme applicative Java.
- Les beans doivent implémenter deux types d'interfaces :
  -une interface home, chargée de créer et gérer les beans.
  -une interface "normale" qui implémente les méthodes métiers des beans.
- Ces interfaces peuvent être soient locales soit distantes.

V.6. Composants entreprise java beans :

- Dans une application J2EE multi-tiers, le tiers EJB héberge la partie de l'application concernant le logique métier et fournit les services système ainsi que la gestion de la concurrence, des transactions et de la sécurité.
- La technologie EJB fournit un modèle de composants distribués qui permet aux développeurs de se concentrer sur les problèmes liés au logique métier de l'application, en se reposant sur la plate-forme J2EE pour toutes les interactions complexes avec le système.
- Dans le modèle de programmation J2EE les composants EJB sont le lien fondamental entre les composants de présentation hébergés par le tiers web et les données de l'application stockées sur les serveurs du système d'information de l'entreprise.

V.7. Visibilité des EJB :

Au-delà de la simple délimitation des différentes couches applicatives, les EJB définissent la manière dont interagissent les différents intervenants d'une architecture Java EE.

Ils définissent, de plus, les possibilités offertes aux divers clients (application Java, applet, application s'exécutant sur le même serveur d'applications) et les Modalités de communication.

Ainsi, il sera possible de définir des EJB suivant deux perspectives pour le client : une vue locale (local) et une vue distante (Remote). **[31]**

V.7.1. Visibilité locale :

En adoptant une vue locale (local) pour un EJB, tout client exécuté dans la même machine virtuelle (autre EJB, servlet...) est en mesure d'appeler les méthodes de cet EJB.

Dans cette vue, les appels de méthode de l'EJB par le client sont effectués comme dans toute application Java classique (Java SE). Les arguments sont passés par référence et il est possible, pour le client de modifier directement les objets récupérés. Il en résulte une démarcation plus faible de l'EJB vis-à-vis de ses clients. Il faut alors envisager que différents clients manient le même objet au même moment et donc anticiper les effets indésirables que cela peut induire.

En revanche, l'utilisation d'une vue locale permet d'optimiser les performances du serveur d'applications et de minimiser les ressources. Celui-ci n'a alors pas à s'occuper des spécificités liées au transport via le réseau (pas de sérialisation des objets, aucune communication réseau...) **[31]**

V.7.2. Visibilité distante :

En adoptant une vue distante (Remote), un EJB met à disposition ses méthodes à des Clients s'exécutant sur des machines virtuelles différentes, et donc sur des machines physiques différentes (applets, applications Java, etc.).

Dans le cadre d'une vue distante, les démarcations sont plus fortes entre un EJB et son client. Les appels de méthodes se font via la technologie RMI, les arguments et valeurs de retour doivent être sérialisés et ne se transmettent plus par référence. Il n'est alors plus possible qu'un client modifie le même objet d'un autre client, et il est donc plus aisé de délimiter les différents domaines de sécurité.

Par contre, l'utilisation d'une vue distante a aussi des inconvénients. Les objets devant être «transportables» à distance, le conteneur doit sérialiser/désérialiser ces objets pour les transmettre via le réseau. Il en résulte des temps de traitements plus élevés par rapport aux appels locaux. **[31]**

V.7.3. Visibilité service web :

Les services web se répandent de plus en plus sur Internet, parce qu'ils permettent d'utiliser n'importe quel service à partir de n'importe quel langage.

Il est possible de spécifier la visibilité de votre EJB avec le type web service pour qu'il puisse être utilisé à la manière d'un service web. Toutefois, ce choix se restreint aux EJB de type Stateless Session Bean. Cette technologie manque cependant de maturité et doit pour le moment rester au niveau des préoccupations de veille technologique. **[31]**

V.8. Les diffèrent type d'EJB :

Il existe deux types d'EJB : les beans de session (session beans) et les beans entité (entity beans). Depuis la version 2.0 des EJB, il existe un troisième type de bean : les beans orienté message (message driven beans). Ces trois types de bean possèdent des points communs notamment celui de devoir être déployés dans un conteneur d'EJB.

Les sessions beans peuvent être de deux types : sans état (stateless) ou avec état (stateful). Les beans de session sans état peuvent être utilisés pour traiter les requêtes de plusieurs clients. Les beans de session avec état ne sont accessibles que lors d'un ou plusieurs échanges avec le même client. Ce type de bean peut conserver des données entre les échanges avec le client. **[25]**

Les beans entité assurent la persistance des données. Il existe deux types d'entity bean:

- persistance gérée par le conteneur (CMP).

-persistance gérée par le bean (BMP).

Avec un bean entité CMP, c'est le conteneur d'EJB qui assure la persistance des données. Un bean entité BMP, assure lui-même la persistance des données grâce à du code inclus dans le bean.

V.9. Les type de Bean :

Il existe 3 types de Bean Entreprise :

V.9.1. Les EJB session :

Un EJB session est un EJB de service dont la durée de vie correspond à un échange avec un client. Ils contiennent les règles métiers de l'application.

Les EJB session stateful sont capables de conserver l'état du bean dans des variables d'instance durant toute la conversation avec un client. Mais ces données ne sont pas persistantes : à la fin de l'échange avec le client, l'instance de l'EJB est détruite et les données sont perdues. Les EJB session stateless ne peuvent pas conserver de telles données entre chaque appel du client.

Il ne faut pas faire appel directement aux méthodes create () et remove () de l'EJB. C'est le conteneur d'EJB qui se charge de la gestion du cycle de vie de l'EJB et qui appelle ces

méthodes. Le client décide simplement du moment de la création et de la suppression du bean en passant par le conteneur.

Une classe qui encapsule un EJB session doit implémenter l'interface javax.ejb.SessionBean elle ne doit pas implémenter les interfaces home et Remote mais elle doit définir les méthodes déclarées dans ces deux interfaces. La classe qui implémente le bean doit définir les méthodes définies dans l'interface Remote.

La classe doit aussi définir la méthode ejbCreate (), ejbRemove (), ejbActivate (), ejbPassivate et setSessionContext ().

La méthode ejbRemove () est appelée par le conteneur lors de la suppression de l'instance du bean.

Pour permettre au serveur d'application d'assurer la montée en charge des différentes applications qui s'exécutent dans ces conteneurs, celui-ci peut momentanément libérer de la mémoire en déchargeant un ou plusieurs beans. Cette action consiste à sérialiser le bean sur le système de fichiers et à le déssérialiser pour sa remontée en mémoire. Lors de ces deux actions, le conteneur appelle respectivement les méthodes ejbPassivate () et ejbActivate ().

La particularité principale d'un Stateful Session Bean est de conserver son état entre différents appels de méthodes. **[34]**

V.9.2. Les EJB entité :

Les EJB entité sont des beans ayant majoritairement pour vocation d'être persistants, c'est-à-dire pouvant être stockés sur un support physique entre deux sessions.

Les EJB BMP sont des beans dont la persistance a dû être programmée par le développeur, ce dernier doit respecter un format pour la classe et les méthodes à implémenter sont imposées par la norme.

Les EJB CMP sont eux des beans dont la persistance est directement assurée par le conteneur d'EJB , le mapping entre l'objet et son support de persistance est indiqué au conteneur via les fichiers descripteurs de déploiement. Le développeur une fois le fichier de déploiement réalisé n'a pas besoin d'écrire le code de persistance.

Depuis la version 3.0 de la spécification EJB, la notion de bean BMP/CMP n'existe plus : les EJB entité sont directement liés à la base de données via un mapping objet relationnel. Ce mapping est défini soit dans un fichier de configuration XML, ou directement

dans le code Java en utilisant des annotations. Cette nouvelle interface de programmation des EJB entité est appelée Java Persistance API. [34]

V.9.3. Les EJB message :

Depuis la norme EJB 2.0, cette architecture propose un troisième type de composant : les EJB message permettant de déclencher un processus côté serveur applicatif lors de la publication d'un message asynchrone. Pour ces composants, le client ne s'adresse pas directement aux composants mais publie un message sur un réceptacle JMS (queue ou topic) configuré sur le serveur applicatif qui va alors déclencher l'activation par ce serveur d'une instance de l'EJB concerné pour pouvoir traiter ce message. [34]

V.10. Création d'un EJB :

V.10.1. Conteneur d'EJB :

Le conteneur d'EJB est un environnement d'exécution qui contient et fait fonctionner les composants EJB tout en leur fournissant un ensemble de services. Les responsabilités des conteneurs d'EJB sont définies précisément par la spécification pour permettre une neutralité vis-à-vis du fournisseur. Les conteneurs d'EJB fournissent la machinerie bas-niveau des EJBs, incluant les transactions distribuées, la sécurité, la gestion du cycle de vie des beans, la mise en cache, la gestion de la concurrence et des sessions.

Le fournisseur de conteneur d'EJB est responsable de la mise à disposition d'un conteneur d'EJB. Il existe plusieurs conteneurs d'EJB commerciaux mais aussi d'excellent conteneur d'EJB open source notamment gratuit Glassfish, JBoss ou Jonas. [35]

V.10.2. Serveur EJB :

Un serveur d'EJB est défini comme un Serveur d'Applications et comporte un ou plusieurs conteneurs d'EJBs. Le fournisseur de serveur EJB est responsable de la mise à disposition d'un serveur EJB. Vous pouvez généralement considérer que le conteneur d'EJB et le serveur EJB sont une seule et même chose. [35]

V.10.3. Java Naming and Directory Interface JNDI :

Il est utilisé dans les Entreprise Java Beans comme le service de nommage pour les composants EJB sur le réseau et pour les autres services du conteneur comme les transactions. JNDI ressemble fort aux autres standards de nommage et de répertoires tels que CORBA CosNaming et peut être implémenté comme un adaptateur de celui-ci. [35]

V.11. Les composants d'un EJB :

V.11.1. Entreprise Bean :

L'Enterprise Bean est une classe Java que le fournisseur d'Enterprise Bean développe. Elle implémente une interface Enterprise Bean (pour plus de détails, voir la section qui suit) et fournit l'implémentation des méthodes métier que le composant supporte. La classe n'implémente aucun mécanisme d'autorisation ou d'authentification, de concurrence ou transactionnel. **[35]**

V.11.2. Interface Home :

Chaque Enterprise Bean créé doit être associé à une interface Home. L'interface Home est utilisée comme une Factory de votre EJB. Les clients utilisent l'interface Home pour trouver une instance de votre EJB ou pour créer une nouvelle instance de votre EJB. **[35]**

V.11.3. Interface Remote :

L'interface Remote est l'interface Java qui reflète les méthodes de l'Enterprise Bean que l'on souhaite rendre disponible au monde extérieur. L'interface Remote joue un rôle similaire à celui de l'interface IDL de CORBA. **[35]**

V.11.4. Descripteur de Déploiement :

Le descripteur de Déploiement est un fichier XML qui contient les informations relatives à l'EJB. L'utilisation d'XML permet au Déployeur de facilement changer les attributs propres à l'EJB. Les attributs configurables définis dans le descripteur de déploiement incluent:

- Les noms des interfaces Home et Remote que nécessite l'EJB.
- Le nom avec lequel sera publiée dans JNDI l'interface Home de l'EJB.
- Les attributs transactionnels pour chaque méthode de l'EJB.

Les listes de contrôle d'accès (Access Control Lists) pour l'authentification. **[35]**

V.11.5. Fichier EJB-Jar :

Le fichier EJB-Jar est un fichier jar Java normal qui contient un EJB, les interfaces Home et Remote ainsi que le descripteur de déploiement.

Maintenant que l'on a un fichier EJB-Jar contenant un Bean, les interfaces Home et Remote, et un descripteur de déploiement, voyons un peu comment toutes ces pièces vont

ensemble, pourquoi les interfaces Home et Remote sont nécessaires et comment le conteneur d'EJB les utilise.

Le conteneur d'EJB implémente les interfaces Home et Remote qui sont dans le fichier EJBJar. Comme mentionné précédemment, l'interface Home met à disposition les méthodes pour créer et trouver votre EJB. Cela signifie que le conteneur d'EJB est responsable de la gestion du cycle de vie de votre EJB. Ce niveau d'abstraction permet aux optimisations d'intervenir. Par exemple, cinq clients peuvent demander simultanément la création d'un EJB à travers l'interface Home, le conteneur d'EJB pourrait n'en créer qu'une seule et partager cet EJB entre les cinq clients. Ceci est réalisé à travers l'interface Remote, qui est aussi implémentée par le conteneur d'EJB. L'objet implémentant Remote joue le rôle d'objet proxy vers l'EJB.

Tous les appels de l'EJB sont redirigés à travers le conteneur d'EJB grâce aux interfaces Home et Remote. Cette abstraction explique aussi pourquoi le conteneur d'EJB peut contrôler la sécurité et le comportement transactionnel. **[35]**

V.12. EJB 2.0 :

V.12.1. Session Bean :

La spécification EJB 2 impose le développement de plusieurs éléments : la classe du Bean, les interfaces et la déclaration du Bean dans le descripteur de déploiement. **[36]**

V.12.1.1. Méthodes du cycle de vie :

Les méthodes du cycle de vie permettent au développeur d'effectuer des opérations lorsque l'état de l'instance du Bean est modifié par le conteneur. Les différentes méthodes de gestion du cycle de vie pour un Session Bean sont:

- EjbActivate ().
- EjbPassivate ().
- EjbRemove ().
- EjbCreate ().
- SetSessionContext (javax.ejb.SessionContext).

Ces méthodes sont appelées automatiquement par le conteneur à des moments précis.

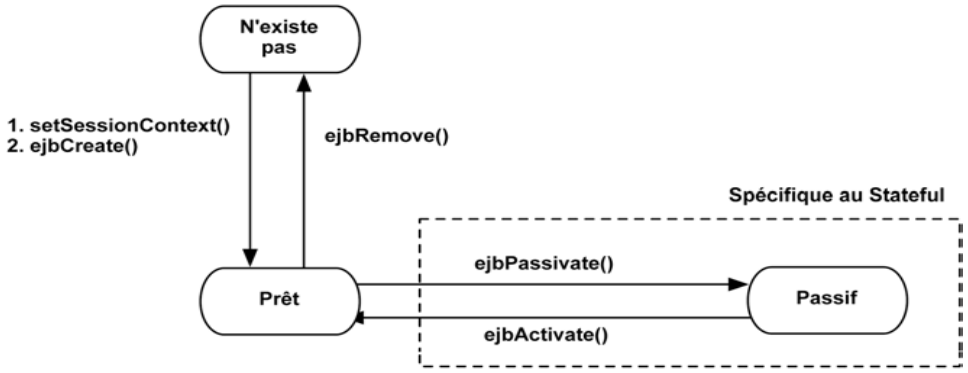

Figure 16 : Cycle de vie d'une session Bean

V.12.1.2. Les interface :

Les interfaces d'un EJB permettent au client d'avoir connaissance des méthodes qu'il a à sa disposition, sans être dépendant de la classe de celles-ci. Avec EJB 2, en plus de définir le code de l'EJB, il est nécessaire de réaliser au moins deux interfaces. La première définit les méthodes de création de l'EJB, c'est l'interface de fabrication home. La seconde regroupe les méthodes métiers disponibles pour le client, c'est l'interface métier business.

Il existe, cependant, plusieurs types de clients: les locaux (local) et les distants (remote). Ces deux types d'interfaces, home et business, se déclinent donc, elles aussi, pour ces deux types d'accès. Il existera donc quatre interfaces distinctes:

- Local home.
- Home (pour l'accès « remote home »).
- Local (pour l'accès « local business »).
- Remote (pour l'accès « remote business »).

V.12.1.3. Le descripteur de déploiement :

Après avoir réalisé les classes des Beans et les interfaces correspondantes, il faut déclarer le Session Bean dans le descripteur de déploiement EJB « ejb-jar.xml », Le conteneur est alors en mesure d'initialiser l'ensemble des EJB déclarés ainsi que les services associés.

V.12.2. Entity bean CMP :

Nous ne parlons cependant que le cas des CMP où la persistance est gérée automatiquement par le conteneur. Il existe, en effet un mode BMP qui permet d'écrire « manuellement » toute cette gestion. Un Entity Bean CMP n'est pas une simple classe Java et impose la création d'un ensemble d'éléments : une classe, les interfaces fabrique (home) et métier (business), et une déclaration dans le descripteur de déploiement « ejb-jar.xml ». **[36]**

V.12.2.1. La classe du Bean :

Dans le modèle CMP 2.x, l'état des Entity Beans est automatiquement géré par le conteneur. La responsabilité de générer le code lié à l'Entity Bean, lors du déploiement. Le développeur, quant à lui, doit décrire les attributs et les relations par des champs persistants et relationnels virtuels. Ils sont qualifiés de virtuels car le développeur ne déclare pas ces champs directement mais utilise des accesseurs (getters and setters) abstraits qui doivent être déclarés dans la classe du Bean. C'est à partir de cette classe que le conteneur génère la classe d'implémentation. Cette phase varie en fonction des techniques employées par le conteneur.

Les méthodes ejbLoad (), ejbStore () et ejbRemove () ont pour vocation d'interagir avec la base de données. Elles permettent respectivement de charger, d'enregistrer les données, et de supprimer l'enregistrement. Ces méthodes sont peu utilisées en général car le conteneur s'occupe de gérer la synchronisation avec la base de données.

Elles s'avèrent, cependant, intéressantes lorsque vous souhaitez effectuer des opérations avant la mise à jour de la base de données (ejbStore()) ou après le chargement des données (ejbLoad()). Imaginons que l'Entity Bean utilise des champs binaires ou de texte à capacité importante, il sera peut-être intéressant de (dé)compressé ces données. On peut alors utiliser ces méthodes pour effectuer des opérations.

Les méthodes ejbPassivate () et ejbActivate () servent à écrire sur disque l'état de l'entité persistante lorsqu'elle n'est plus utilisée depuis un certain temps. Ces méthodes sont peu utilisées dans la pratique car le conteneur gère très bien cela, dans la plupart des cas.

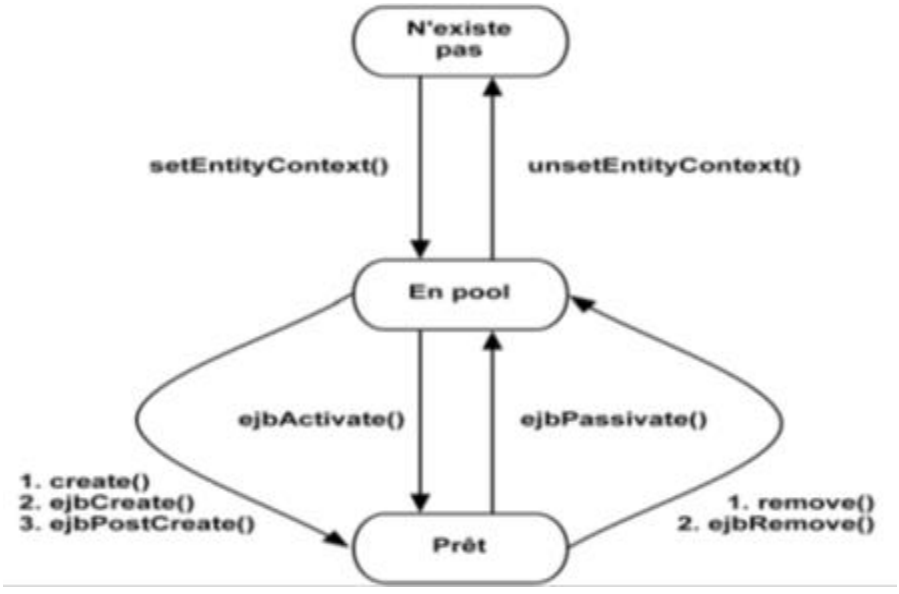

Figure 17 : Cycle de vie d'un Entity Bean.

Les méthodes setEntityContext () et un setEntityContext () permettent d'informer le Bean lors de sa création et de sa destruction. Lors de l'appel de setEntityContext (), le développeur doit se charger de récupérer les ressources qui ne sont pas spécifiques à l'Entity Bean (fabrique de connexion JMS, data source…).L'appel de un setEntityContext () indique la fin de vie du Bean, c'est le bon moment pour libérer les ressources chargées lors de l'appel de setEntityContext ().

Les méthodes ejbCreate () et ejbPostCreate () sont obligatoires, sans pour autant être définies dans l'interface EntityBean. Elles servent à définir la façon de créer une nouvelle instance de l'Entity Bean.

Il est possible de surcharger la méthode ejbCreate () si l'on souhaite avoir différentes façons de créer une entité. Le type de retour de la méthode ejbCreate () doit être le type de la clé primaire. Néanmoins, pour chaque méthode ejbCreate () présente, une méthode ejbPostCreate () ayant les mêmes arguments doit exister. La méthode ejbCreate () est appelée par le conteneur lorsqu'une application cliente souhaite créer une nouvelle entité persistante.

Cette méthode doit se charger d'affecter (via les setters) les valeurs pour chaque champ persistant. Une fois cette méthode exécutée, le conteneur appelle la méthode ejbPostCreate () correspondante qui s'occupe, quant à elle, d'affecter les champs relationnels.

V.12.2.2. Message Driven Bean :

La classe du MDB: Tout Message Driven Bean doit implémenter l'interface javax.ejb.MessageDriven-Bean. Cette interface définit les méthodes liées à son intégration dans le conteneur EJB. Elle contient deux méthodes.

SetMessageDrivenContext: cette méthode est appelée après l'instanciation du MDB. Elle lui permet de conserver une référence au contexte lié au conteneur pour ensuite agir avec le conteneur EJB.

EjbRemove () : cette méthode est appelée avant l'arrêt du MDB. Ceci se produit quand le conteneur juge nécessaire de retirer l'instance de son pool (lors de la baisse du nombre de requêtes ou de la fermeture de l'application).

Dans le cadre d'utilisation de JMS, il est obligatoire d'implémenter l'interface javax.jms.MessageListener. Elle peut être assimilée à l'interface métier d'un Session Bean.

V.13. Les inconvénients des EJBs 2 :

Les EJB 2 ont tellement fait parler d'eux, qu'ils ont terni l'image de J2EE. Même si certaines entreprises utilisaient EJB 2 pour ses points forts (gestion intégrée des transactions…), beaucoup d'autres avaient alors à l'esprit la lourdeur de ces EJB, l'assimilant

à J2EE, ce qui est loin d'être le cas. Des problèmes récurrents sont de plus retournés par les développeurs:

Entity Bean: la spécification est une erreur conceptuelle, elle est trop restrictive. Le langage EJB-QL apporte cependant une « rustine » aux problèmes de flexibilité, mais reste trop limité.

Mauvaise productivité : Le développement d'un EJB nécessite beaucoup trop de dépendances (trop d'interfaces, longueur des descripteurs de déploiement…)

Complexité de développement: obligation d'utiliser des outils externes (XDoclet, ejbgen…) et de mettre en place des design patterns (Service Locator, Value Object, Business Interface…). Il est également difficile d'effectuer des tests unitaires (exécution dans le conteneur EJB seulement).

La communauté développe des Framework « maisons » qui se sont très vite répandus sur le marché. Ces Framework, appelés « conteneurs légers », ont l'avantage d'être plus performants qu'un conteneur lourd de type EJB 2, même s'ils ne supportent pas toute l'architecture distribuée des EJB.

La nouvelle spécification EJB 3 est publier en mai 2006, après deux ans d'étude, que la refonte complète de la spécification J2EE, et plus particulièrement des EJB, est réalisée. C'est le début de Java EE 5 et des EJB 3, tout en gardant une compatibilité ascendante sur J2EE 1.4 et EJB 2.

V.14. EJB 3.0 :

La spécification EJB 3 est le résultat d'un ensemble d'évolutions et d'innovations qui se sont installées sur le marché sans pour autant faire partie intégrante de la spécification Java EE 5. De nombreux Framework et technologies ont approuvé certains principes et modèles de développement. La spécification a su prendre parti des avantages de chacune de ces innovations afin d'élaborer la meilleure structure possible pour les EJB.

Pièce maîtresse de l'architecture JEE, les EJB 3 apportent des modifications notables dans le mode de développement et intègrent de nombreuses nouveautés, notamment en ce qui concerne la persistance. Le passage des EJB 2.1 en 3.0 apporte une simplification radicale en supprimant les descripteurs de déploiement, les appels JNDI, et laisse place aux annotations.

Il existe deux grandes familles d'EJB qui s'exécutant en mode asynchrone: entité et session, et message Driven qui s'exécute en mode synchrone. **[36]**

V.14.1 Session bean:

L'API a été grandement simplifiée en EJB 3. Dorénavant, il n'est plus nécessaire de maintenir différentes interfaces suivant le type d'accès que l'on souhaite donner au client.Le développeur n'a plus à créer les interfaces home et peut se concentrer sur l'interface business. Le couplage avec les interfaces de la spécification n'est plus nécessaire. Les éléments à mettre en place pour un Session Bean sont la classe du Bean et l'interface business (métier). Désormais, une seule interface est nécessaire, et la déclaration au sein du descripteur de déploiement n'est plus obligatoire.

V.14.1.1 La classe du Bean:

La classe d'un Session Bean EJB 3 n'a pas à implémenter d'interface, comme c'est le cas avec un Session Bean EJB 2. Il suffit ici d'utiliser les annotations @Remote ou @Local pour définir respectivement si le Session Bean fournit les méthodes à des clients distants ou locaux. Il en est de même pour les types de Session Beans, grâce aux annotations @Stateless ou @Stateful. Contrairement à un Session Bean EJB 2, la classe du Session Bean EJB 3 implémente directement la ou les interfaces métiers utilisées. Cela élimine toutes les erreurs d'inattention que les développeurs pouvaient faire (oubli d'implémentation de telle méthode métier définie dans l'interface…). La classe ne contient plus directement l'implémentation des méthodes du cycle de vie (setSessionContext (), ejbActivate…). Néanmoins, il est toujours possible de spécifier des méthodes qui seront appelées lors du changement d'état dans le cycle de vie.

V.14.1.2 Les méthodes du cycle de vie :

Afin de gérer la vie d'un Session Bean, le conteneur EJB doit procéder à différentes étapes, qui constituent le cycle de vie de celui-ci. Tout d'abord, pour exploiter un Session Bean, celui-ci doit être instancié. De façon très simplifiée, le conteneur s'occupe de cette instanciation par l'appel de la méthode NewInstance (). Celle-ci est disponible à partir de l'objet Class lié à la classe du Session Bean, L'intérêt de « passivation » est de libérer la mémoire employée par le Stateful inutilisé pour le moment. Le conteneur peut alors sérialiser l'instance sur un support externe.

@PostActivate (Stateful uniquement) qui joue le rôle inverse de @PrePassivate. Cette annotation permet de spécifier la méthode qui sera appelée par le conteneur lorsqu'un Bean devra être réactivé de son état de passivation. Le Bean devra retrouver un état opérationnel en récupérant les ressources éventuellement libérées lors de la «passivation » On parle d'«activation».

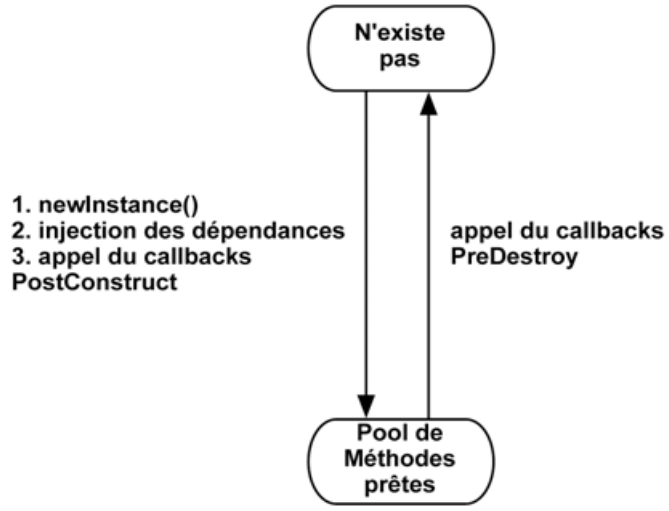

Figure 18 : Cycle de vie d'un Stateless Session Bean (EJB 3).

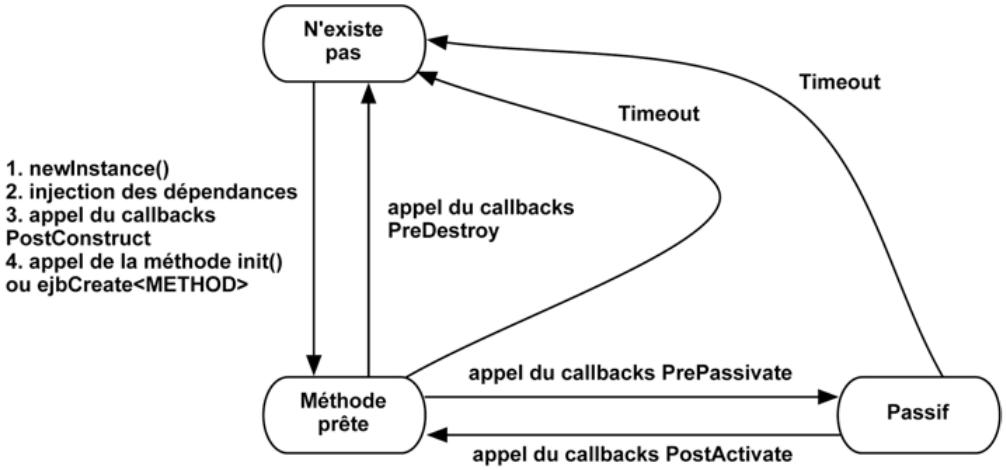

Figure 19 : Cycle de vie d'un Stateful Session Bean (EJB 3).

L'état « passivé » est réservé aux Stateful Session Beans car ce sont les seuls à partager un état avec le client. Le cycle de vie pour ce type de composant commence au premier appel d'une méthode métier. Le conteneur crée alors une instance via la méthode Class.newInstance(). Il injecte alors les dépendances et appelle les méthodes annotées avec @PostConstruct. La grande différence avec le type Stateless est que le Stateful intègre le concept de « passivation ». L'implémentation de ce concept est spécifique au fournisseur de conteneur et peut donc varier d'un serveur à l'autre. Dans tous les cas, le conteneur appelle les méthodes annotées avec @PrePassivate et @PostActivate au moment de la passivation et de l'activation du composant. L'instance est supprimée lorsque le client appelle une méthode annotée avec @Remove ou lorsque le temps imparti pour la session est dépassé (timeout).

Il existe deux possibilités de déclarer ces méthodes callback interceptos :

La première solution est, de définir des méthodes callback directement dans la classe du composant. Dans ce cas, la signature des méthodes doit respecter la syntaxe suivante, où <METHOD> correspond à un nom de méthode choisi.

La seconde solution consiste en la séparation des méthodes de gestion du cycle de vie dans des classes intercepteurs.

V.14.1.3. Les interface métiers :

Avec EJB 3, les interfaces métiers sont dites des « pures ».C'est-à-dire qu'elles n'ont plus besoin d'hériter de javax.ejb.EJBObject ou de javax.ejb.EJBLocalObject. Les méthodes de cette interface doivent cependant répondre aux contraintes suivantes :

- Leur nom est libre tant qu'il ne commence pas par « ejb » pour éviter tout conflit avec les méthodes callback interceptos.
- Elles doivent être déclarées comme étant public.
- Elles ne doivent pas être déclarées comme final ou static.
- Dans le cas d'un accès de type remote, le type de retour et le type des arguments doivent être compatibles avec des transports de type RMI/IIOP.
- De même, dans le cas d'un service web, le type de retour et des arguments doivent être compatibles avec un transport de type JAX-WS/JAX-RPC

Les méthodes des interfaces métiers peuvent désormais lever des exceptions liées à l'application, via la clause throws. Il n'est de même plus nécessaire de spécifier pour les méthodes de l'interface distante, l'exception java.rmi.RemoteException.

Si un problème survient au niveau du protocole, une exception de type javax.ejb.EJBException est levée par le conteneur. Celle-ci étant de type « runtime », le développeur n'a pas à la spécifier dans l'élaboration de ses interfaces.

V.14.2. Entity Bean :

La spécification EJB3 apporte de nombreux changements et simplifications pour le développement d'Entity Beans. Le changement principal est l'externalisation de la gestion de la persistance avec l'API Persistance 1.0 (JSR 220). Cette API fournit un modèle de persistance à base de POJO pour le mapping objet/relationnel. La simplicité de développement est en partie due à l'introduction des annotations et à la configuration par exception. En effet les spécifications définissent des comportements par défaut qui sont généralement corrects pour une majeure partie des applications. L'autre grande nouveauté est

la suppression des interfaces liées aux Entity Beans. Même si cela impose l'utilisation d'un Session Bean pour gérer l'accès aux données, les architectes s'en féliciteront ! En effet, il n'est pas possible d'accéder directement à l'Entity Bean depuis l'application cliente.

Cette évolution est, avant tout, le fruit de la communauté. C'est par l'essor des Framework, tels Hibernate, iBatis ou Toplink que le mapping objet/relationnel a pu se développer et finalement devenir un standard.

V.14.2.1. La classe de l'entité :

La classe de l'Entity Bean est le premier élément à définir en EJB 3. Les Entity Beans sont des POJO. De ce fait, la création d'un Entity Bean User se résume à la création d'une classe User. L'état persistant de l'entité est représenté par les variables d'instances de la classe qui correspondent aux propriétés du POJO. Ces variables peuvent être privées, protégées ou non spécifiées. Les clients ne peuvent alors pas accède directement aux variables et doivent utiliser les accesseurs (getter and setter) ou d'autres méthodes métiers de la classe. Cette classe doit, cependant, respecter certaines règles:

- Cette classe peut être abstraite ou concrète.
- La classe peut aussi bien hériter d'une classe entité que d'une classe non entité, et inversement.
- Les méthodes, les propriétés de la classe et la classe elle-même ne doivent pas être finales.
- Si une instance de l'entité a la possibilité d'être envoyée à un client distant, alors la classe doit implémenter java.io.Serializable. En effet, RMI utilise la (dé)sérialisation pour passer les arguments entre l'application cliente et le serveur.
- La classe doit posséder un constructeur sans argument qui peut être public ou protégé.

Néanmoins, la classe peut posséder des constructeurs surchargés si la condition précédente est respectée.

Les Entity Beans EJB3 supportent l'héritage, les associations et les requêtes polymorphiques. C'est grâce à l'annotation @Entity que le conteneur pourra savoir quelles classes il doit considérer en tant qu'Entity Bean. Cette annotation se situe au niveau de la classe et permet de définir si besoin le nom de l'entité via l'attribut name. Le nom utilisé doit être unique dans une application.

V.14.3. Message driven bean :

Les précédents types d'EJB offrent des services de manière synchrone. Le client émet une requête, puis attend que l'EJB lui envoie un résultat. Pour les MDB, le comportement est complètement différent. Les clients n'appellent pas directement des méthodes mais, utilisent JMS pour produire un message et le publier dans une file d'attente.

À l'autre bout, le MDB est à l'écoute de cette file d'attente et se « réveille » à l'arrivée d'un message. Il extrait ce dernier de la file d'attente, en récupère le contenu puis exécute un traitement. Le client n'a donc pas besoin de figer son exécution durant le traitement du MDB. Le traitement est asynchrone.

V.14.3.1. La classe du Bean :

En EJB3, il n'est maintenant plus obligatoire d'implémenter l'interface MessageDrivenBean. Seule l'annotation @MessageDriven est nécessaire pour définir votre classe en tant que MDB. L'implémentation de MessageListener est quant à elle nécessaire si le MDB travaille avec JMS. Il n'est désormais plus obligatoire d'utiliser le descripteur de déploiement pour configurer le MDB. Tout est faisable via les annotations. L'annotation @MessageDriven permet de définir le nom du MDB au sein du conteneur (attribut name).L'attribut activationConfig permet de configurer les propriétés du MDB. Il est possible de récupérer le contexte du MDB via l'injection (annotation @Resource). Le type de cet objet est Message Driven Context.

V.14.3.2. Message Driven Context :

Comme pour les Session Beans, les MDB ont également un contexte d'exécution de type Message Driven Context. Il est similaire à l'objet Session Context expliqué dans la partie Session Bean. En effet, cette interface hérite simplement d'EJB Context et ne rajoute aucune méthode.

L'utilisation de cet objet est cependant particulière pour les MDB. Les méthodes getEJBHome () et getEJBLocalHome () lancent une RuntimeException si elles sont invoquées. En effet, un MDB ne possède pas d'interface home. De même pour les méthodes getCallerPrincipal () et isCallerInRole () car un MDB n'est pas appelé par une personne et donc n'a aucun contexte de sécurité associé.

V.15. EJB pour développer des composants business :

- Implémenter du logique métier.

- Accéder à un SGBD.
- Accéder à un autre système d'information (CICS, COBOL, SAP R/3, etc.…).
- Applications web : intégration avec JSP/Servlets.
- Web services basés sur XML (SOAP, UDDI, etc.…)

V.16. Les avantage :

- Le serveur EJB amène des services connus (transaction, sécurité, persistance, cycle de vie, réserve, montée ou diminution de charge, …)

Le client ne connaît que des déclarations des méthodes métiers. Il ne peut manipuler que ce type de méthodes. Il n'a pas accès directement aux implémentations : l'EJB est entièrement géré par le serveur (cycle de vie, lancement des méthodes, association EJB avec le client, …).

- Architecture portable sur de multiples serveurs d'EJB.

Conclusion :

Dans ce chapitre indique-t-on les technologies les plus expérimentés dans la programmation distribuée.

Ce chapitre a été organisé en trois parties : Dans la première partie nous allons parler du CORBA, la deuxième partie concerne les Framework .NET et dans la dernière parité nous allons préciser un peut les composants EJB.

A ce titre EJB amène des services connus (transaction, sécurité, persistance, cycle de vie, réserve, montée ou diminution de charge, …) Dans le chapitre qui suit nous allons présenter la persistance et mapping objet-relationnel.

Chapitre 03 : Conception et Réalisation

Introduction :

Dans ce chapitre nous allons présenter notre application, les étapes nécessaires de la création d'un EJB.

Nous avons choisi la réalisation d'une application de gestion d'une agence touristique, en commençant par la création de la base de donnée avec le model conceptuelle de donnée, puis le développement et le déploiement des composant, enfin la réalisation de l'application client.

I. Système d'information

I .1.Définition :

Un système d'information (SI) est un ensemble organisé de ressources (matériels, logiciels, personnel, données et procédures) qui permet de regrouper, de classifier, de traiter et de diffuser de l'information sur un environnement donné.

L'utilisation de moyens informatiques, électroniques et la télécommunication permettent d'automatiser et de dématérialiser les opérations telles que les procédures d'entreprise surtout en matière logistique. Ils sont aujourd'hui largement utilisés en lieu et place des moyens classiques tels que les formulaires sur papier et le téléphone et cette transformation est à l'origine de la notion de système d'information. **[37]**

I .2.Les phases du SI (cycle de vie) :

I .2.1.Recueil des besoins :

Cette étape contient les activités préalables d'analyse des besoins initiaux et de la rentabilité d'un nouveau système d'information.

I .2.2.Conception :

Les activités de définition d'un cahier des charges, conception, réalisation et mise en œuvre d'un nouveau système d'information.

I .2.3.Exploitation :

Elle contient les activités de choix d'un progiciel en fonction des exigences et contraintes du projet, paramétrage du progiciel, tests des adaptations et mise en œuvre d'un nouveau système d'information. Cette phase se termine lorsque le produit est installé sur les sites des utilisateurs.

I .2.4.Production :

La production contient les activités d'exploitation du système informatique dans son environnement cible ainsi que les services d'assistance offerts aux utilisateurs. Cette phase démarre avant la fin de la phase de développement, avec des activités de préparation de la mise en exploitation. Elle se poursuit jusqu'au retrait du système.

Phase de maintenance/évolution :

Elle contient les activités de gestion des modifications apportées au système d'information pour le faire évoluer et maintenir le système informatique en état de fonctionner. Cette phase se poursuit jusqu'au retrait du système. Les activités de la phase de développement sont mises en œuvre dans le cadre.

I.3.Modélisation de SI :

La conception d'un système d'information (SI) n'est pas évidente car il faut réfléchir à L'ensemble de l'organisation que l'on doit mettre en place. La phase de conception nécessite des méthodes permettant de mettre en place un modèle sur lequel on va s'appuyer. La modélisation consiste à créer une représentation virtuelle d'une réalité de telle façon à faire ressortir les points auxquels on s'intéresse.

Parmi les méthodes approuvées :

- La méthode UML.
- La méthode MERISE.

II. La méthode MERISE :

II.1. Définition :

MERISE (Méthode d'Étude et de Réalisation Informatiques pour des Systèmes d'Entreprise) est une méthode de conception, de développement et de réalisation de projets informatiques. Le but de cette méthode est d'arriver à concevoir un système d'information. La méthode MERISE est basée sur la séparation des données et des traitements à effectuer en plusieurs modèles conceptuels et physiques. C'est d'ailleurs son point fort.

La séparation des données et des traitements assure une longévité au modèle. En effet, l'agencement des données n'a pas à être souvent remanié, tandis que les traitements le sont plus fréquemment.

La méthode MERISE date de 1978-1979, et fait suite à une consultation lancée en 1977 par le ministère Français de l'Industrie dans le but de choisir des sociétés de conseil en informatique afin de définir une méthode de conception de systèmes d'information. Les deux

principales sociétés ayant mis au point cette méthode sont le CTI (Centre Technique d'Informatique) chargé de gérer le projet, et le CETE (Centre d'Études Techniques de l'Équipement) implanté à Aix-en-Provence. **[38]**

Les niveaux d'abstraction de Merise ainsi que ces modèles correspondants sont présentés dans le tableau suivant :

	MODELES		
NIVEAUX	**DONNEES**	**TRAITEMENTS**	**CHOIX PRIX EN COMPTE**
Conceptuel	Modèle Conceptuel des Données (MCD)	Modèle Conceptuel des Traitements (MCT)	**Choix de gestion Quoi ?**
Organisationnel	Modèle Logique des Données (MLD)	Modèle Organisationnel des Traitements (MOT)	**Choix 'organisation Qui ?, Où ?, Quand ?**
Physique	Modèle Physique des Données (MPD)	Modèle Physique des traitements (MPT)	**Choix techniques Comment ?**

Tableau 1 : Les niveaux d'abstraction de merise.

II.2. Le modèle conceptuelle des données (MCD) :

Le modèle conceptuel des données (MCD) a pour but d'écrire de façon formelle les données qui seront utilisées par le système d'information. Il s'agit donc d'une représentation des données, facilement compréhensible, permettant de décrire le système d'information à l'aide d'entités. **[39]**

II.2.1. Les modèles schématiques de représentation :

Le MCD (modèle conceptuel de données) : modèle entité/association souvent nommé entité-relation repose sur les concepts suivants :

- entités
- associations.
- propriétés.

- ***Les entités :*** Une entité est la représentation d'un élément dans un SI.
- ***Les relations :*** une relation ou association est la liaison qui lie entre les entités du SI.
- ***Les propriétés(Attribues)* :** sont les informations de base qui décrivent les éléments(les entités) d'un SI.

II.2.2. MCD vérifié et normalisé :

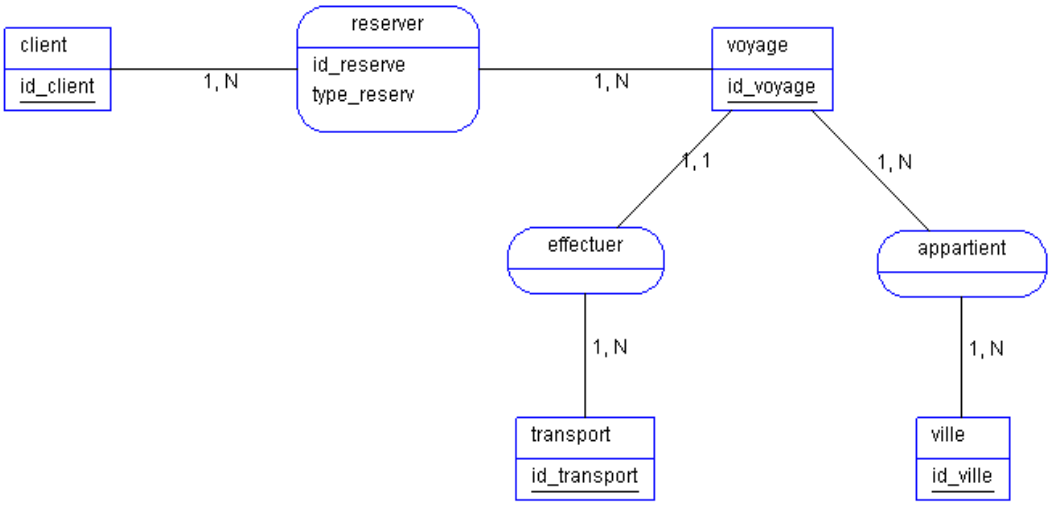

Figure 20 : Modèle conceptuel de données (MCD)

II.3.Le modèle conceptuel des traitements (MCT) :

Le modèle conceptuel des traitements permet de traiter la dynamique du système d'information, c'est-à-dire les opérations qui sont réalisées en fonction d'événements.

Ce modèle permet donc de représenter de façon schématique l'activité d'un système d'information sans faire référence à des choix organisationnels ou des moyens d'exécution, c'est à- dire qu'il permet de définir simplement ce qui doit être fait, mais il ne dit pas quand, comment ni où... **[7]**

II.3.1. Représentation du MCT :

II.3.1.1. Partie client :

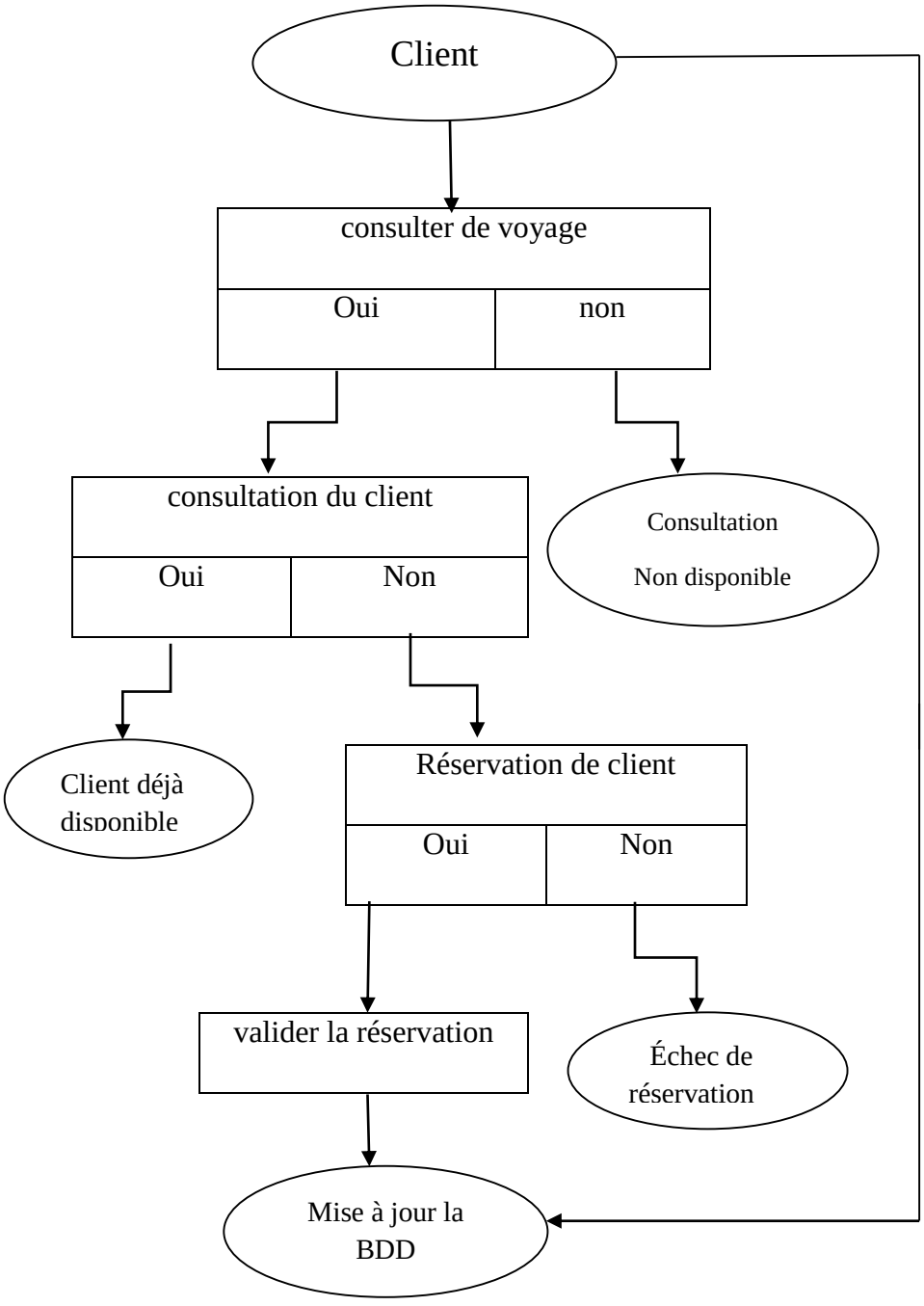

Figure 21 : Le modèle conceptuel des traitements (MCT client)

II.3.1.2. Partie administrateur :

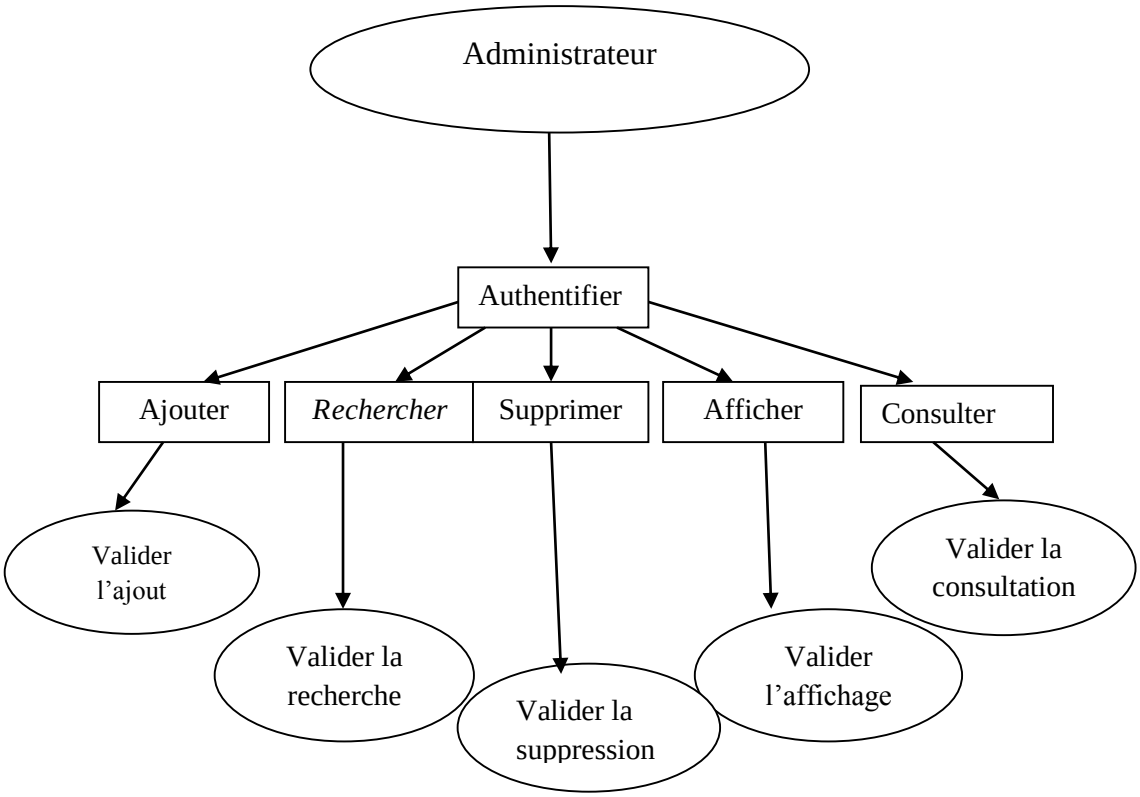

Figure 22 : Le modèle conceptuel des traitements (MCT admin)

II.4. Le modèle logique des données(MLD) :

Le modèle logique de données (MLD) décrit les structures de données indépendamment de la gestion physique des bases de données. Il est une étape intermédiaire, intellectuellement très satisfaisante, vers le modèle physique de données. **[7]**

Le MLD sera comme suit :

- ***Client :*** **(id_client,** nom_client**,** prenom_client, num_passport, adresse, num_tel, code_voyage).
- ***Voyage :*** **(id_voyage,** date_depart, date_arrivee, ville_depart, ville_arrivee, prix_voyage).
- ***Transport*** **: (id_transport,** type_transport**,** prix_transport).
- ***Ville :*** **(id_ville,** nom_ville**,** pays).
- ***Reserve:*** **(id_reserve, id_client, id_voyage,** type_reserve).
- ***Appartient :*** **(id_appartient, id_voyage, id_ville,**)

II.4.1. Élaboration du MLD :

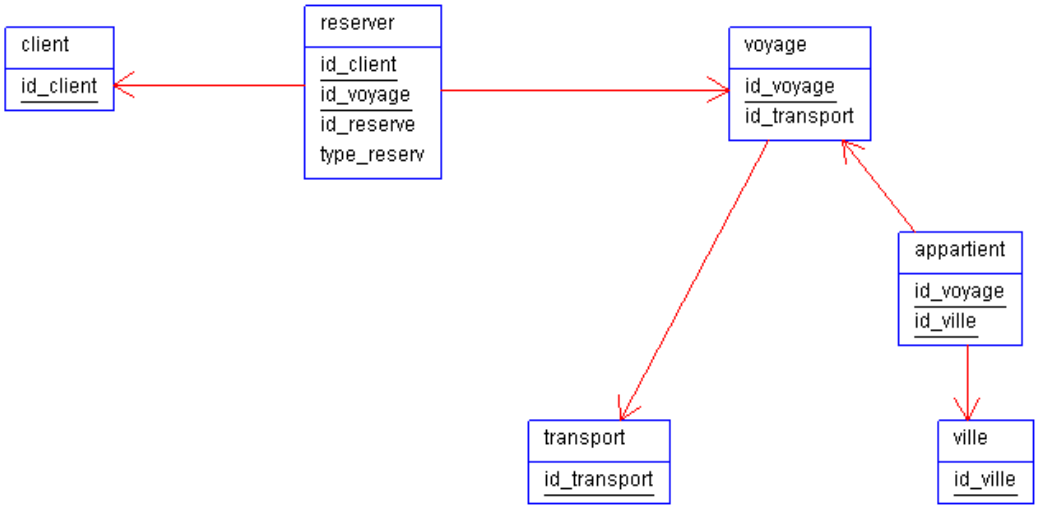

Figure 23 : Le modèle logique des données(MLD)

III. Langage utilisé : JAVA

III.1. Définition :

Le langage Java est un langage généraliste de programmation synthétisant les principaux langages existants lors de sa création en 1995 par *Sun Microsystems*. Il permet une programmation orientée-objet (à l'instar de Small Talk et, dans une moindre mesure, C++), modulaire (langage ADA) et reprend une syntaxe très proche de celle du langage C.

III.2. L'avantage du langage java :

Outre l'orientation objet, le langage Java a l'avantage d'être :

- ***Modulaire*** (on peut écrire des Portions de code génériques, c.-à-d. utilisables par plusieurs applications).
- ***Rigoureux*** (la plupart des erreurs se produisent à la compilation et non à l'exécution),
- ***Portable*** (un même programme compilé peut s'exécuter sur différents environnements).

En contre-partie, les applications Java ont le défaut d'être plus lentes à l'exécution que des applications programmées en C par exemple.

IV.1. Les outils utilisés :

- Le SGBD MySQL 5.0.
- L'IDE NetBeans 6.5.1
- Le serveur d'application Glass Fish V 2.

IV.1.1. MySQL 5.0 :

MySQL est un système de gestion de base de données (SGBD). Selon le type d'application, La licence est libre ou propriétaire. Il fait partie des logiciels de gestion de base de données les Plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle et Microsoft SQL Server.

MySQL est un serveur de bases de données relationnelles SQL développé dans un souci de performances élevées en lecture, ce qui signifie qu'il est d'avantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes et fortement sécurisées. Il est multithread et multiutilisateurs.

IV.1.2. NetBeans 7.2.1 :

NetBeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL (Common Development and Distribution License) et GPLv2. En plus de Java, NetBeans permet également de supporter différents autres langages, comme C, C++, JavaScript, XML, Groovy, PHP et HTML de façon native ainsi que bien d'autres (comme Python ou Ruby) par l'ajout de greffons. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web).

IV.1.3.GlassFish 3.1.2 :

GlassFish V3, l'implémentation de référence de JEE6 (JSR 316), inclut à partir de sa version 3.1 des nouvelles fonctionnalités de clustering (synchronisation lors du démarrage d'une nouvelle instance, réplication des changements dynamiques de configuration,…), et offre également un mécanisme de réplication de la session http et des EJB statefull.V.

V. Les étapes de la réalisation du projet :

À présent, nous allons exposer la partie empirique de notre projet qui regroupe la mise-en-place de notre application gestion d'agence touristique, dont les étapes suivant :

V.1. Création d'une base de données :

Notre base de données sera élaboré avec MySQL 5.0, s'appelle « **base** » et contiendra six tables qui sont illustrées dans la figure suivante :

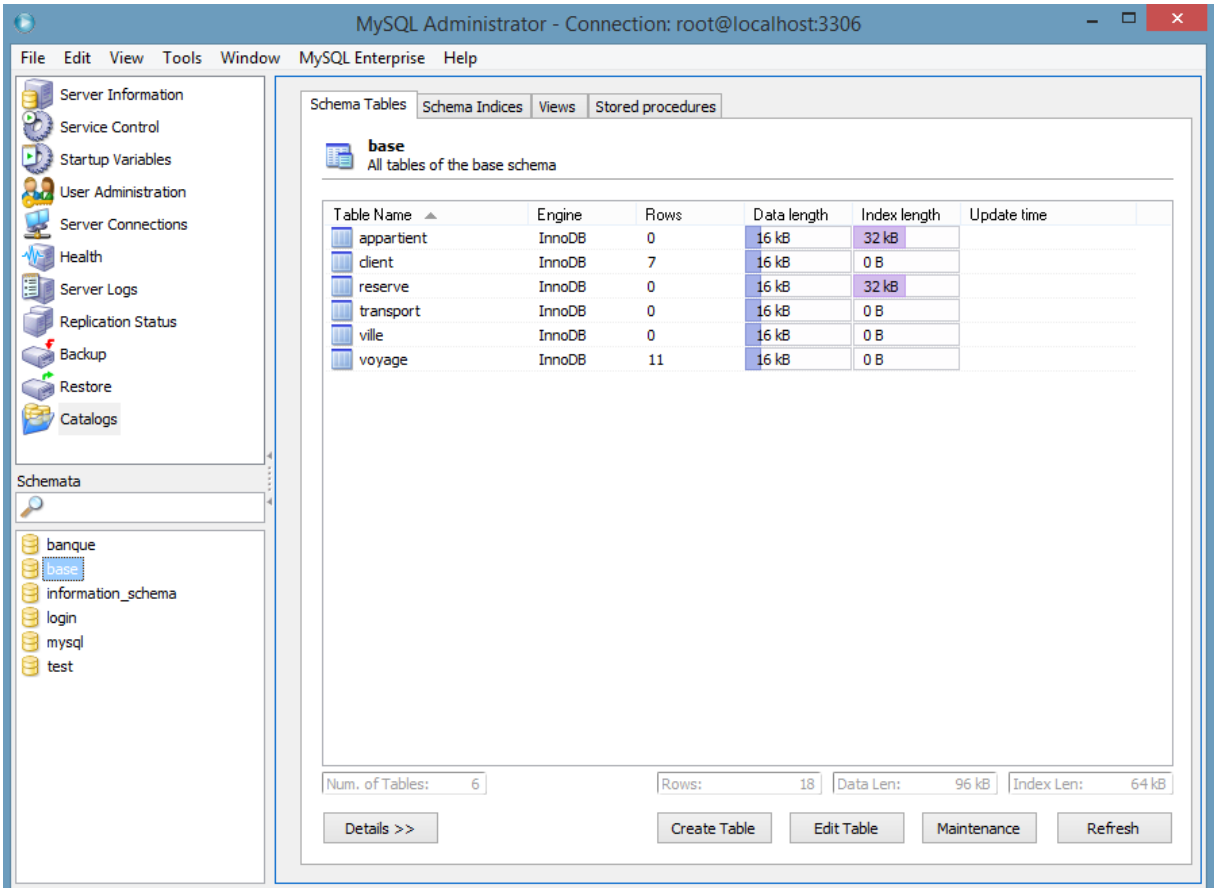

Figure 24 : La base de données.

V.2.Création des composants EJB sous NetBeans :

Après la création de la base de données. Nous nous intéressons maintenant à la définition du schéma général de l'application qui contient trois parties : la base de données (MySQL), les composants EJB (Glass Fish) et l'interface Client.

Dans un serveur d'application il y a une relation entre le Bean session et le Bean Entity, le composant session Bean correspond à un échange avec le client il contient les règles métiers de l'application, l'entité Bean contient la représentation objet de chaque table de la base donnée, il sauvegarde des données traiter par des méthodes de la session Bean, après il sera persisté (stocké) dans une base de données.

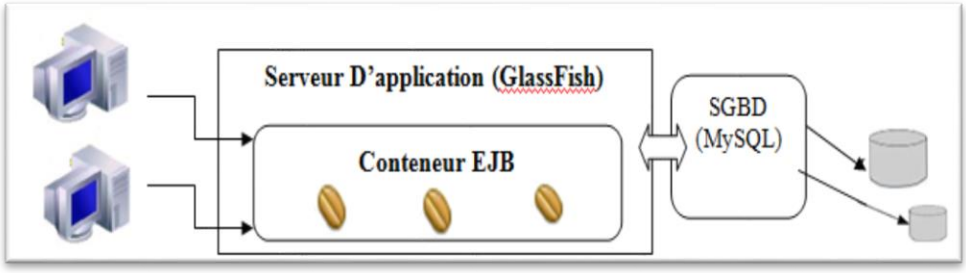

Figure 25 : Schéma général de l'application.

V.2.1.Création d'une class Library :

Premièrement on crée une class Library qui illustré dans le figure suivant, dans cette librairie nous allons créer les composants Entity pour nous permettre l'appel distant (Remote) de ces composants.

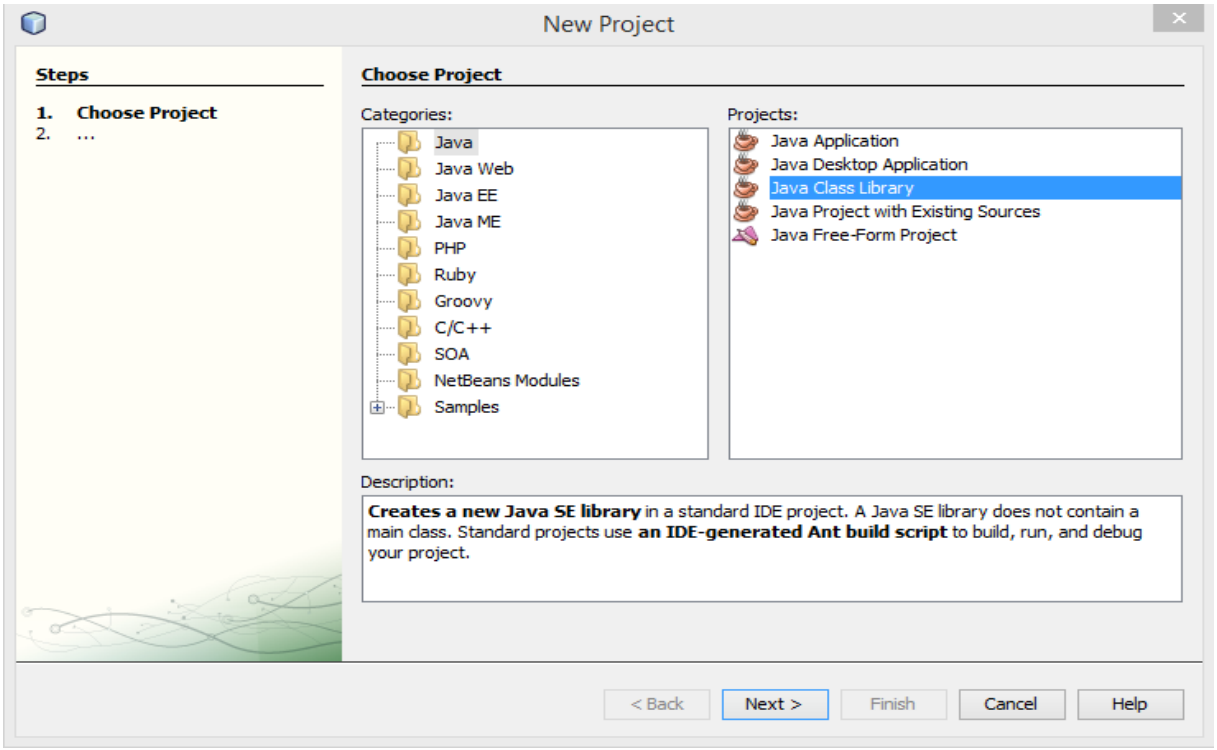

Figure 26 : Création de nouveau projet "Java Class Library".

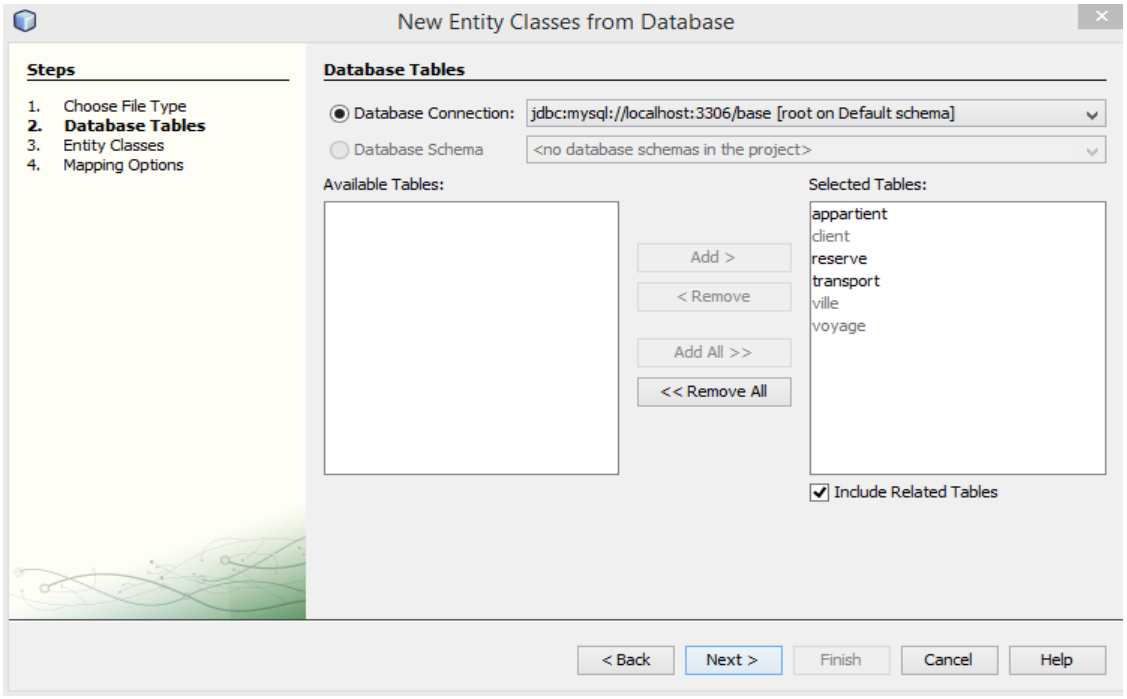

Figure 27 : Création de la connexion et l'ajout d'entité Bean pour chaque table.

V.2.2.Création d'un module EJB :

Le projet EJB Module comprend les Ejbs Session déffinissant l'aspect métier de notre application.

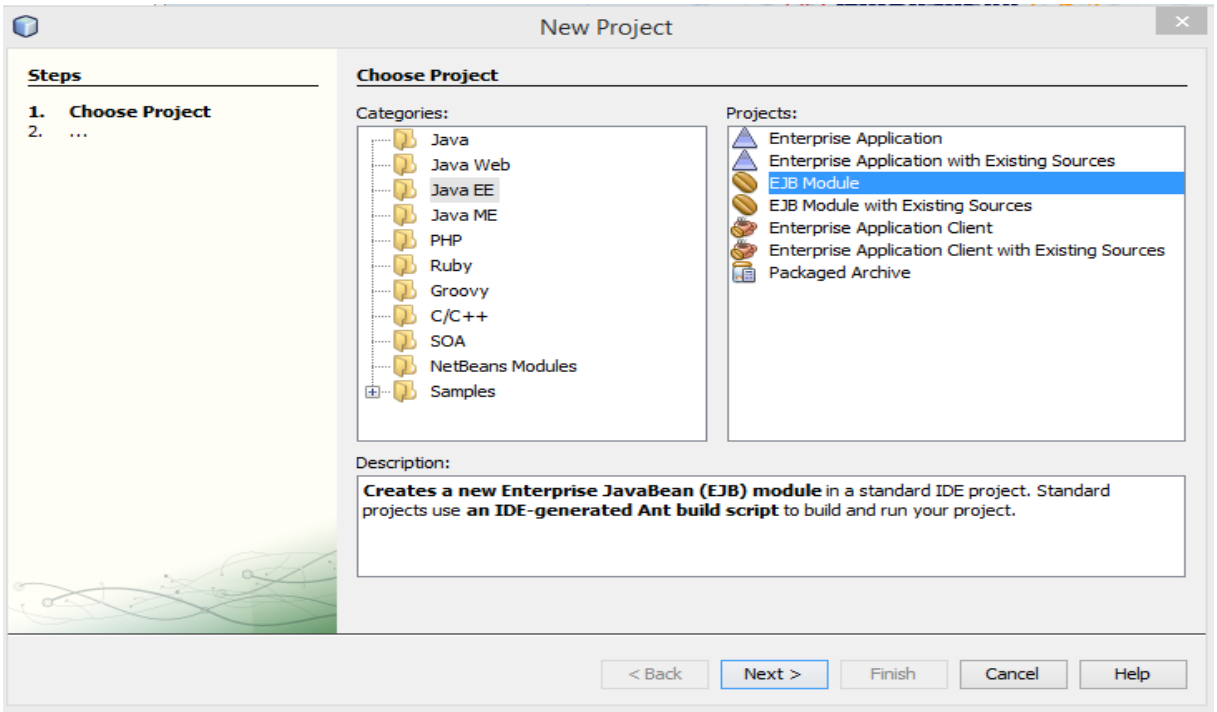

Figure 28 : Création d'un EJB Module.

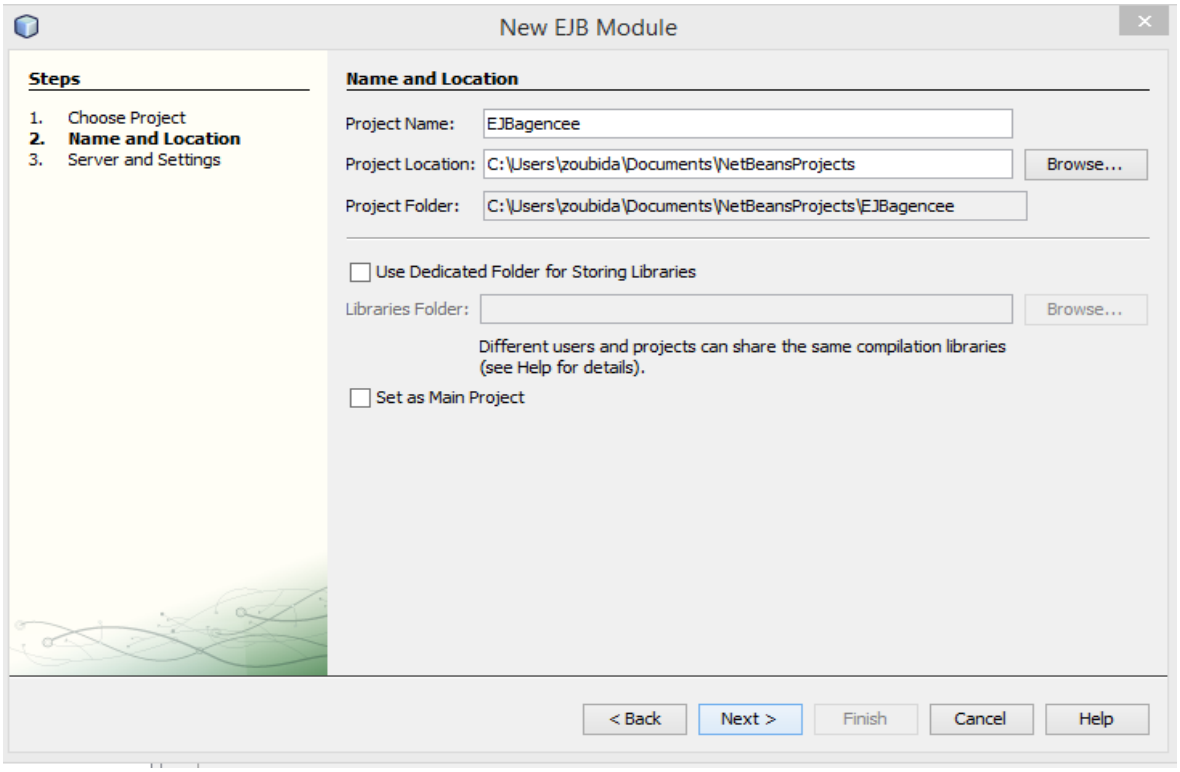

Figure 29 : Création d'un nouveau projet (Nom et place du projet).

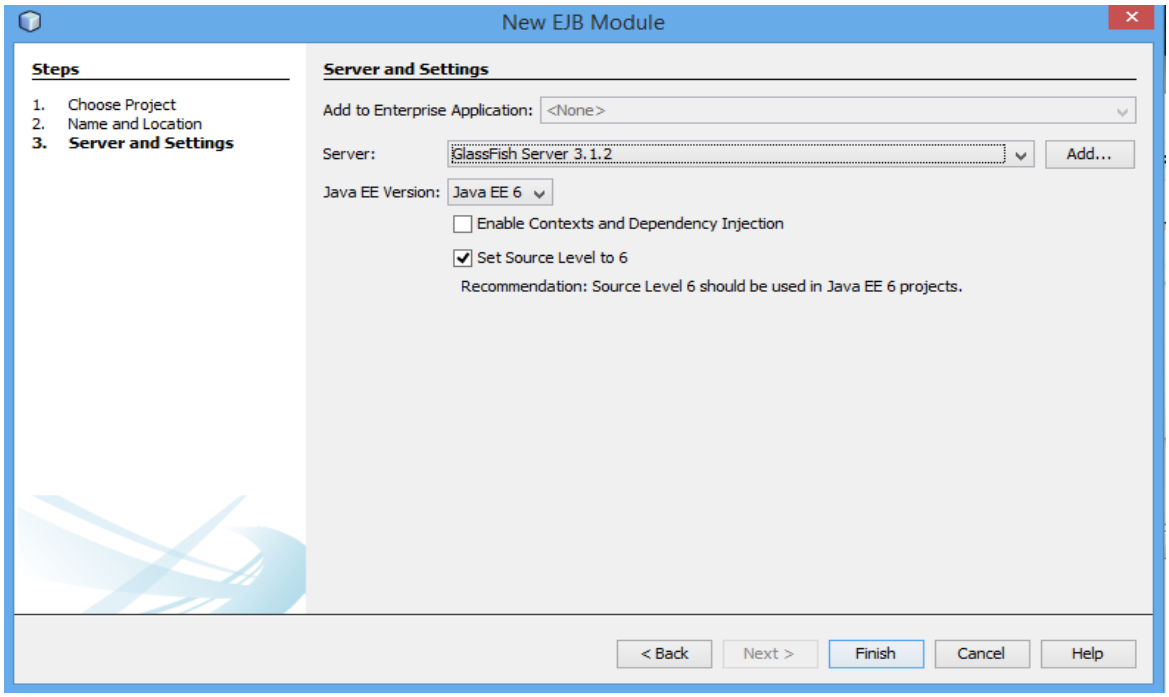

Figure 30 : Création d'un nouveau projet (choix du serveur d'application).

V.2.3. Création des composants sessions pour chaque Entity Bean :

Nous allons créer une session bean pour chaque entité bean (faire Clic Droit sur EJBagence et choisir New->Others).

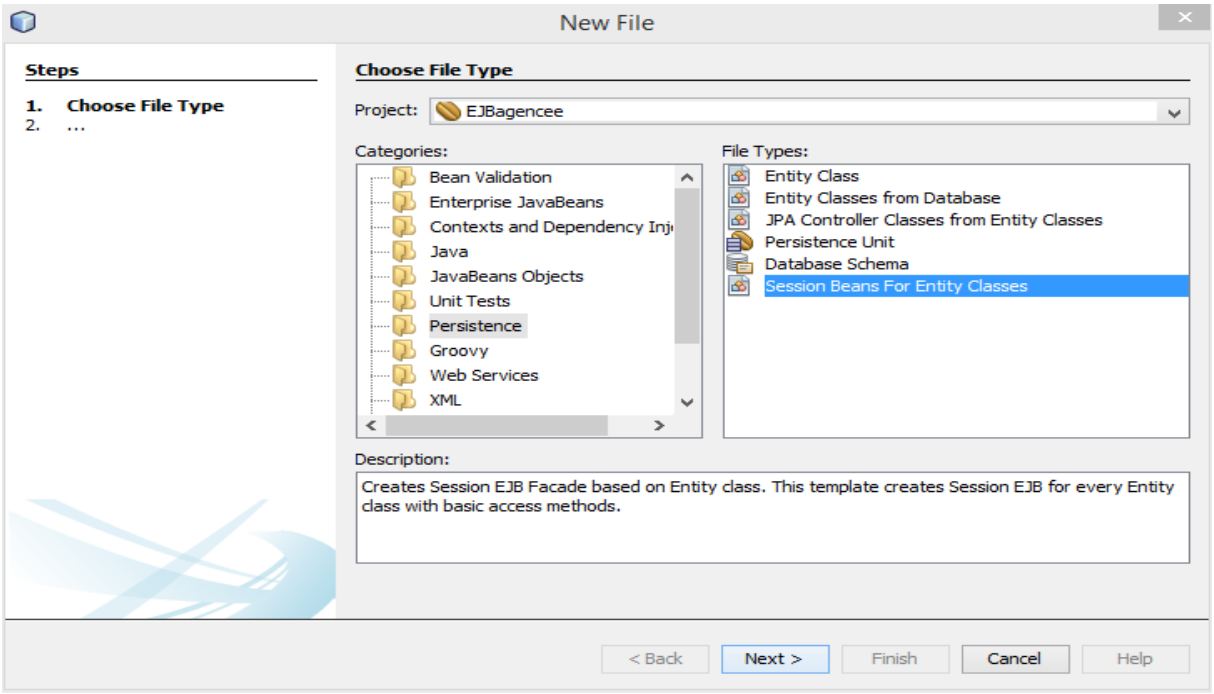

Figure 31 : Création d'une session beans entity classe.

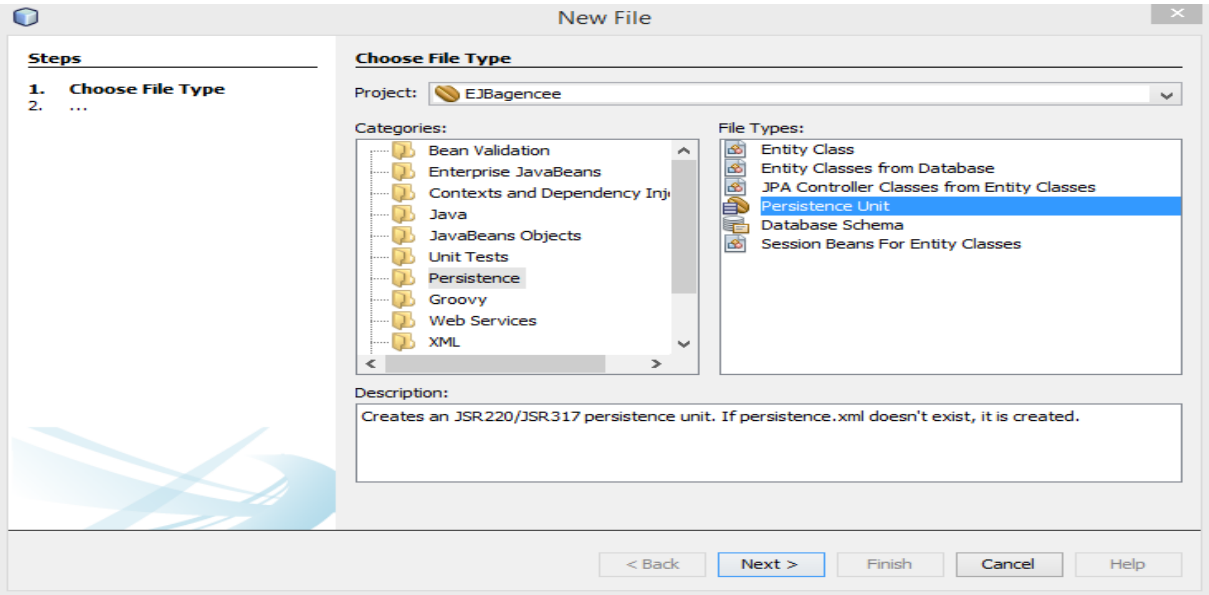

Figure 32 : Création de l'unité de persistance.

Choisir ensuite comme Data Source : base.

Comme fournisseur de service de persistance EclipseLink.

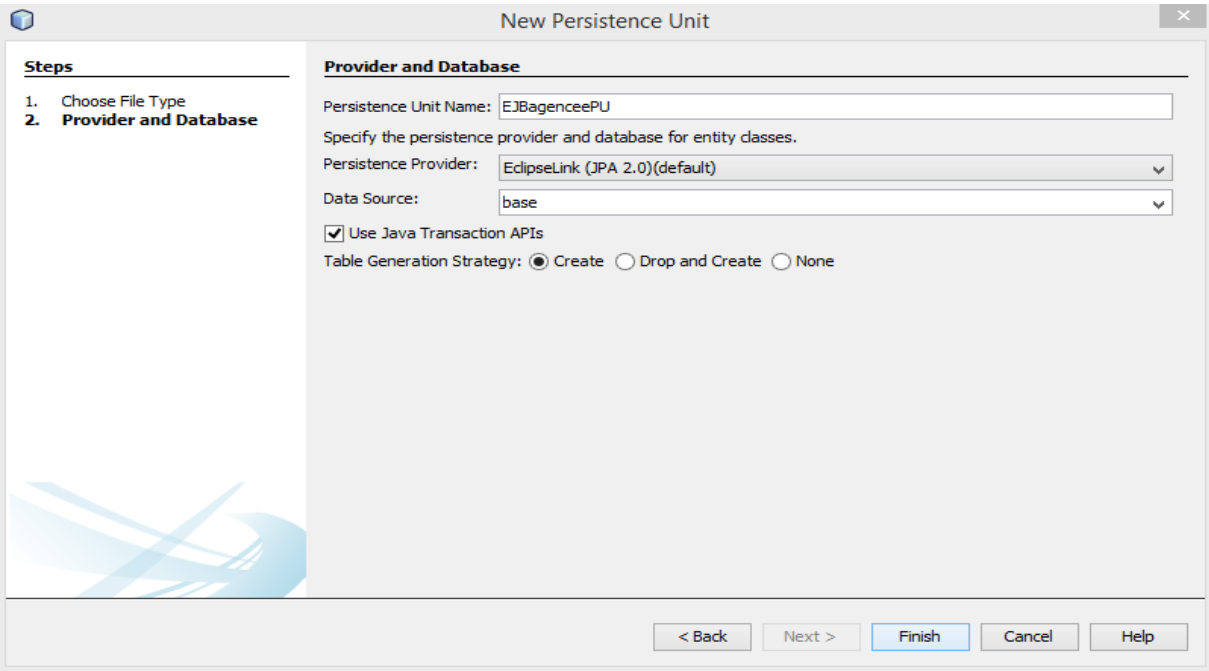

Figure 33 : Choix du fournisseur de la persistance.

Dans la partie « Configuration Files », le fichier **persistence.xml** apparaît

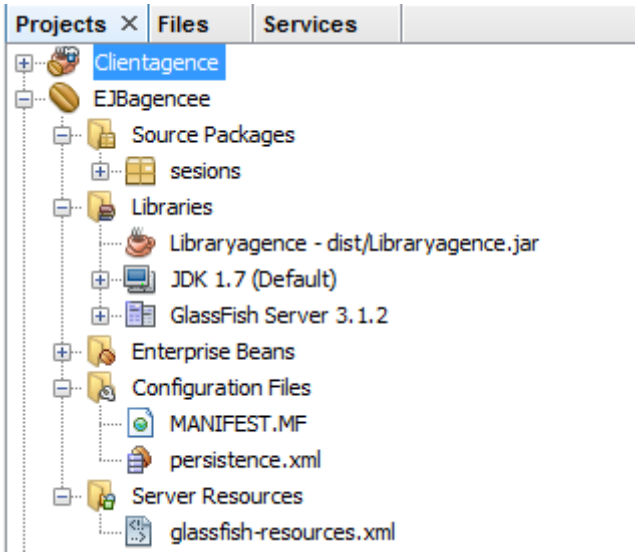

Figure 34 : Fichier de configuration.

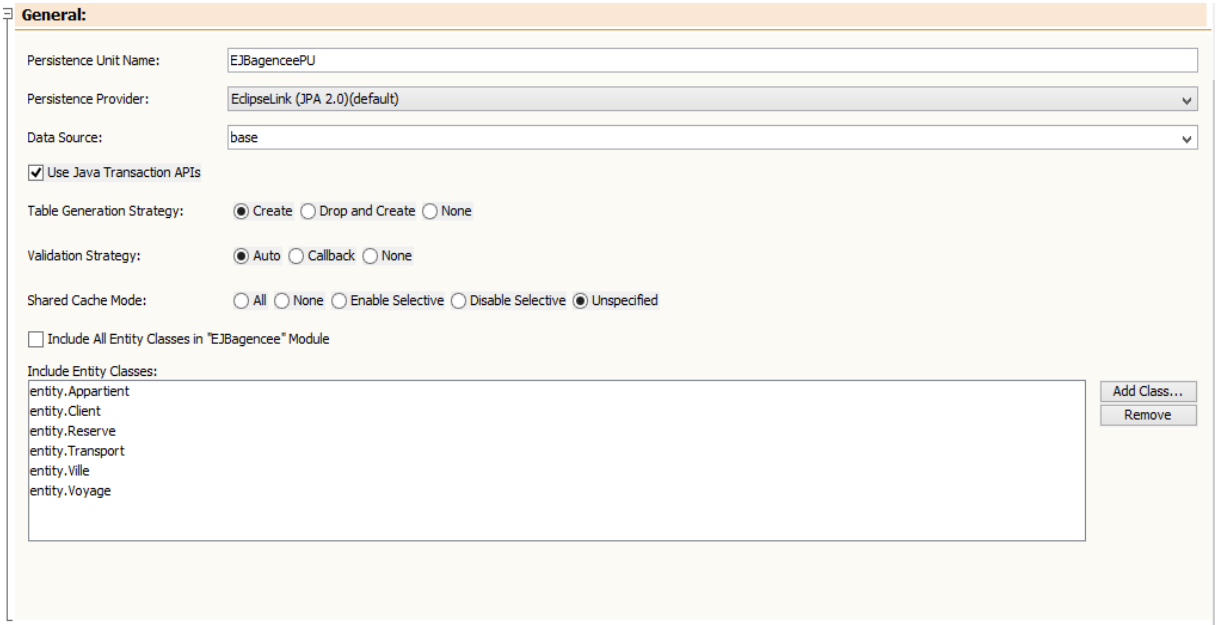

Figure 35 : Configuration de l'unité de persistance.

V.2.4.Déploiement de l'EJB :

Dans cette étape nous allons procéder à la mise en œuvre de nos composants session.

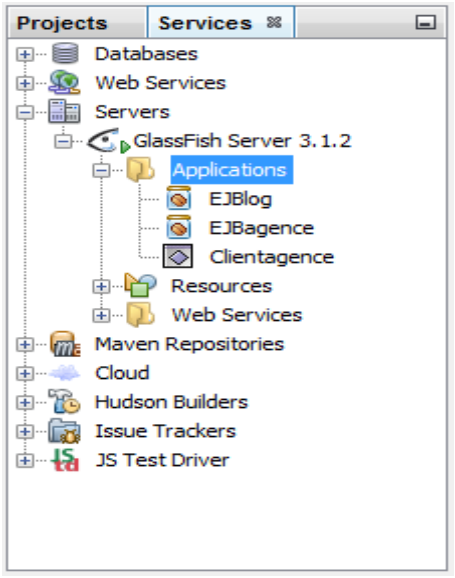

Figure 36 : Déploiement de projet EJBAgence.

VI. Application cliente :

Dans cette partie, nous passons à la concrétisation de l'interface de notre application cliente.

Celui-ci regroupe les services suivants :

- Inscription du client.
- Authentification admin.
- Recherche d'un client.
- Ajout d'un client.
- Affichage d'un client.
- Suppression d'un client.

Après avoir présenté les services de l'application cliente, nous avons exposé la description de quelques fenêtres principales utilisées dans cette dernière.

- Les fenêtres principales de l'interface visuelle.
- Ainsi que quelques aspects techniques liés à la manipulation et l'interrogation des bases de données.

VII. Les Interfaces graphiques :

VII.1.Iinterface générale :

Cette interface représente la page principale qui constitué de six boutons permettent de authentifier, réserver, consulter, contacter, voir les destinations et une aide générale.

Figure 37 : L'interface de l'application.

VII.2. Coté client :

VII.2.1. Réservation :

Il faut remplir les renseignements du client indiqué dans le formulaire dans la figure suivante :

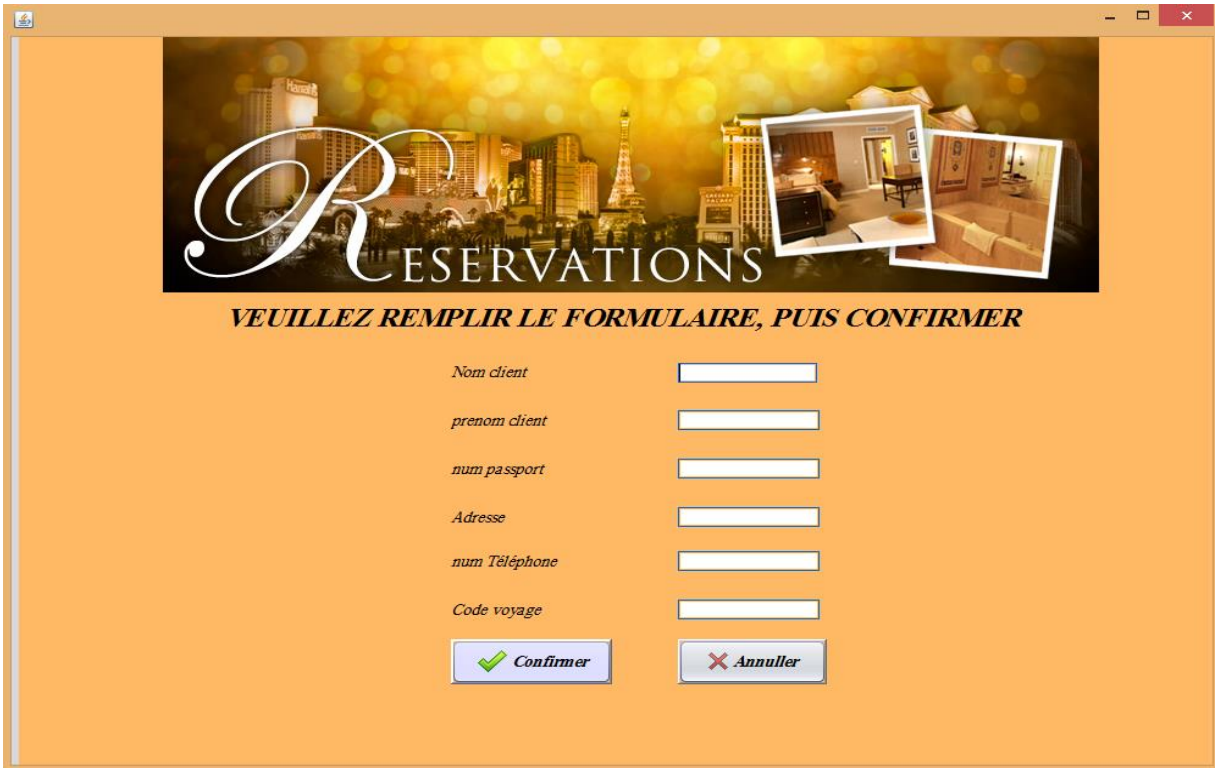

Figure 38 : Réservation de voyage

VII.2.2.Nos destination :

Pour avoir notre destination disponible dans l'agence touristique Ibn Khaldoune.

Figure 39 : Destination de l'agence

VII.2.3.Contacte :

Pour avoir plus d'information contacter nous

Figure 40 : Contacte

VII.2.4.Consultation :

Cette interface permet de diriger le client, elle s'affiche les destinations proposées et leurs dates et l'heure de départ et arrivée.

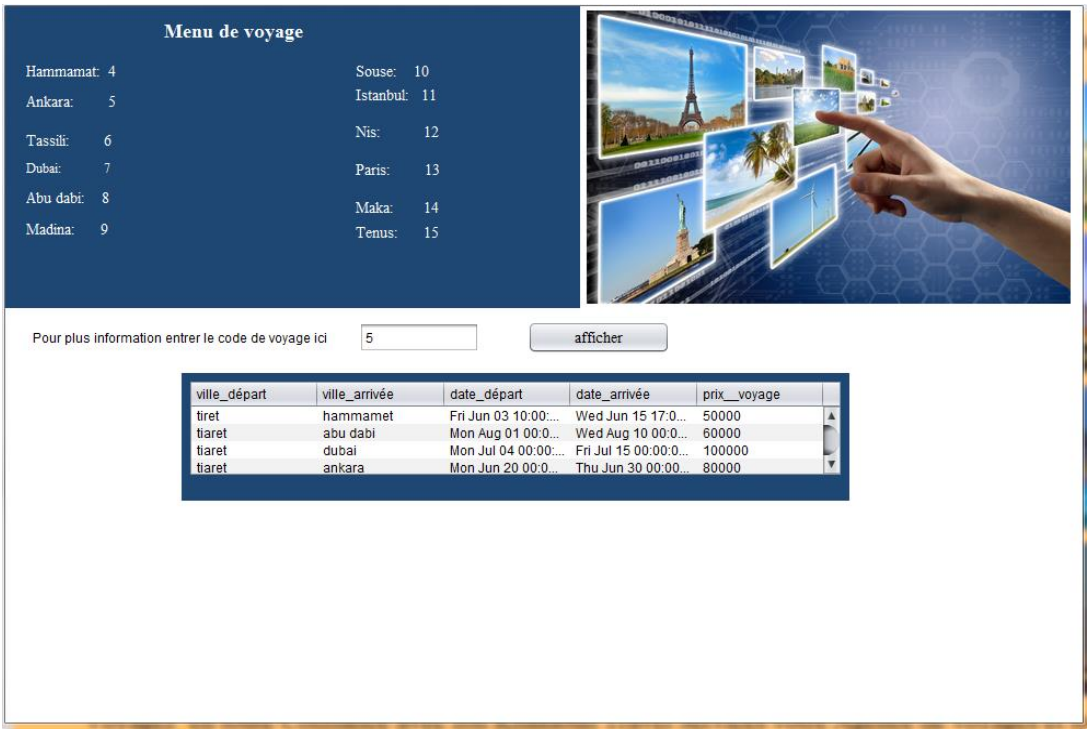

Figure 41 : Consultation des voyages

VII.2.5.Aide :

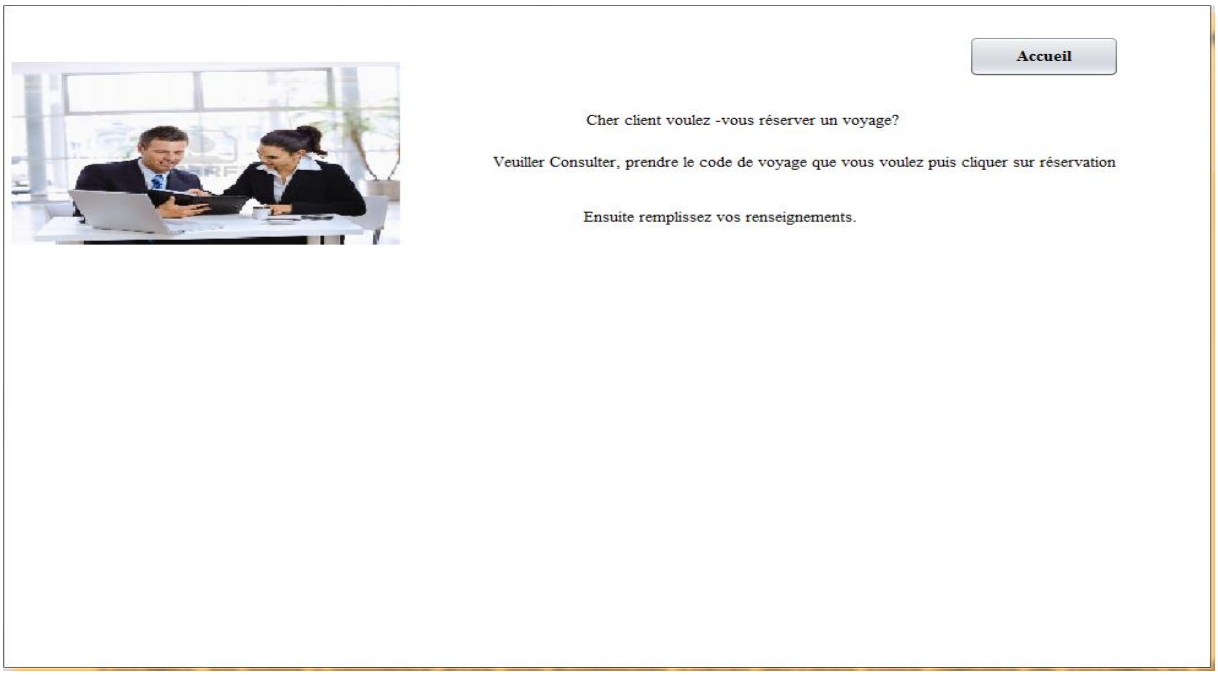

Figure 42 : Aide

V.III. Coté administrateur :

V.III.1.Login :

Cette figure représente le point où l'administrateur doit introduire son nom d'utilisateur et son mot de passe.

Pour cela on a créé une base de données s'appelle « login »qui possède un nom utilisateur et un mot passe, puis on a fait la persistance. Lorsque l'accès est accepté on aura la main à l'utilisation.

Figure 43 : Login de l'administrateur.

V.III.2. Interface de l'administrateur :

V.III.2.1. Mise à jour des voyages :

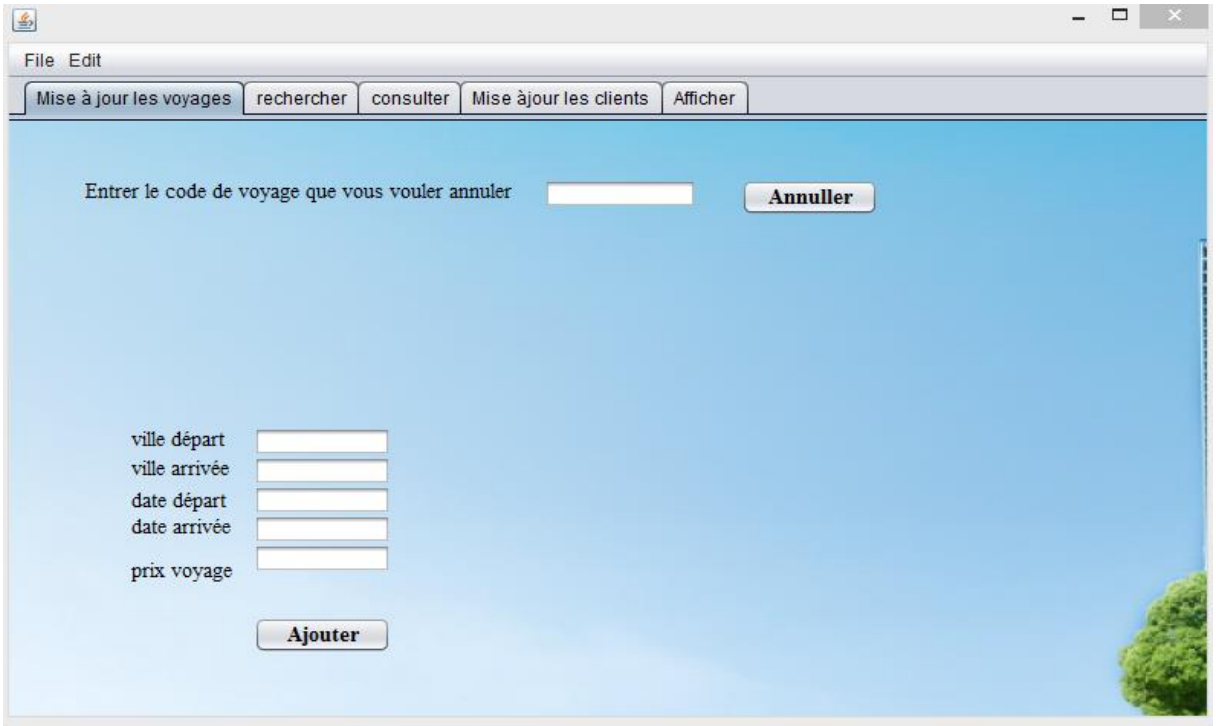

Figure 44 : Mise à jour des voyages

V.III.2.2. Rechercher :

L'administrateur a le droit de rechercher des clients à travers le code du client.

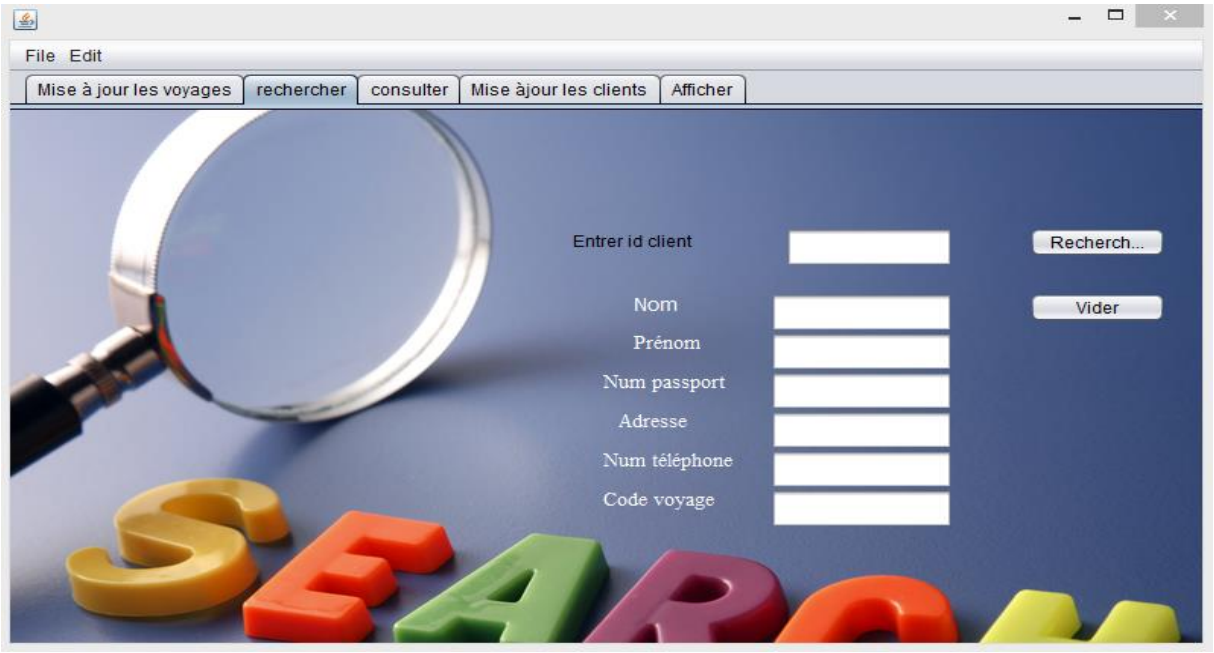

Figure 45 : Fenêtre de la recherche

VIII.2.3.Consulter :

L'administrateur peut afficher le nombre des clients, voyages, les dates des voyages les destinations, etc…

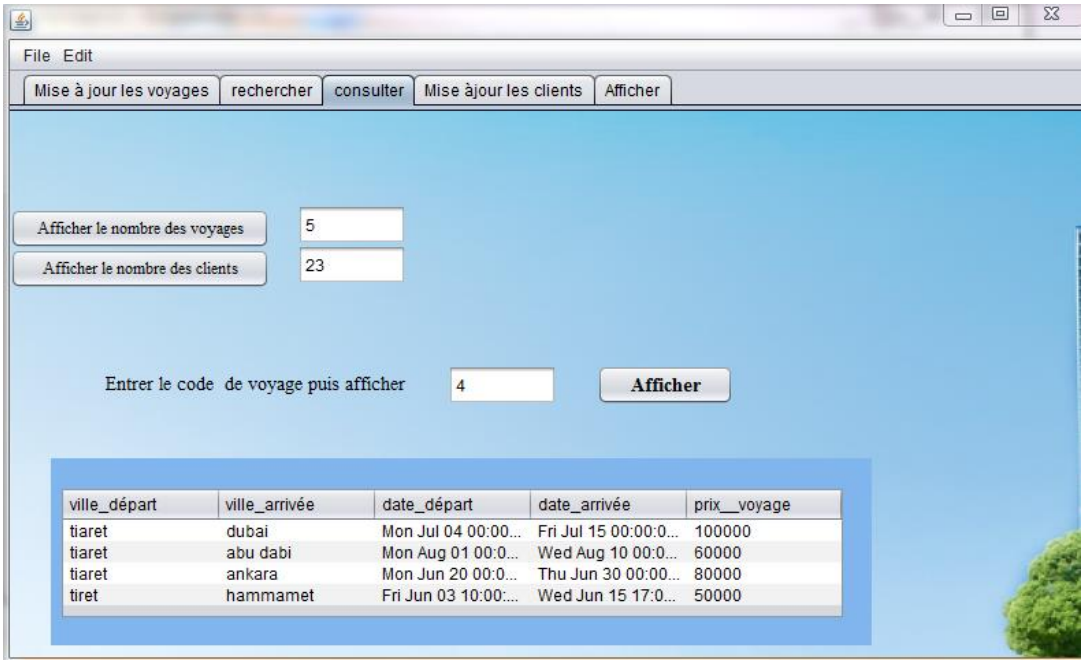

Figure 46 : Fenêtre de consultation administrateur

V.III.2.3. Mise à jour des clients :

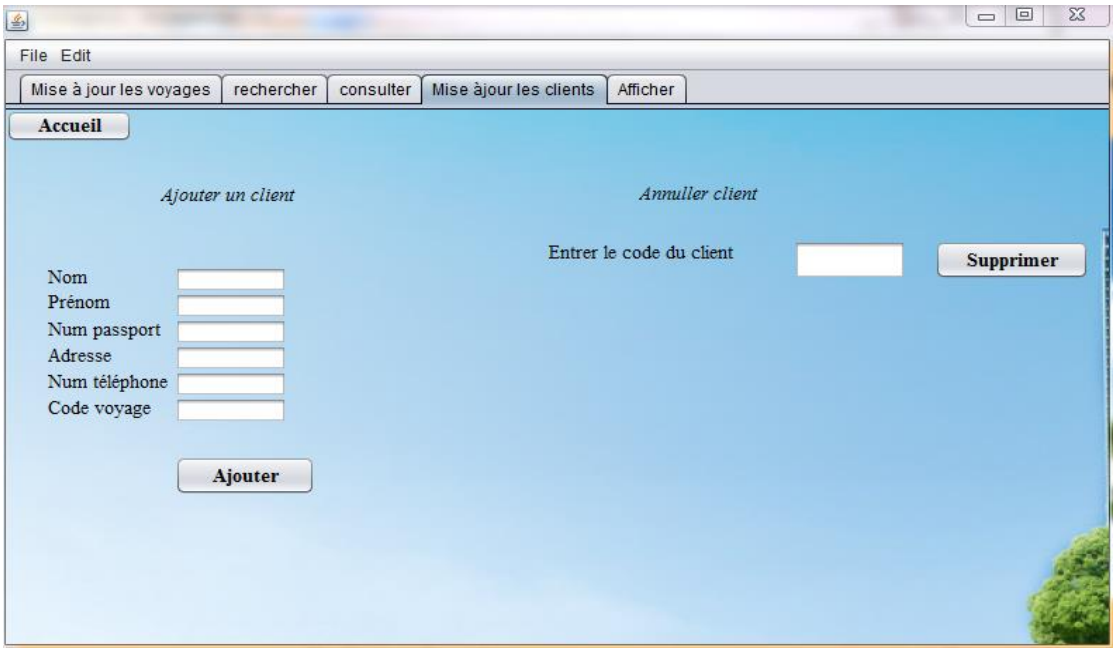

Figure 47 : Mise à jour des clients

V.III.2.4.Afficher :

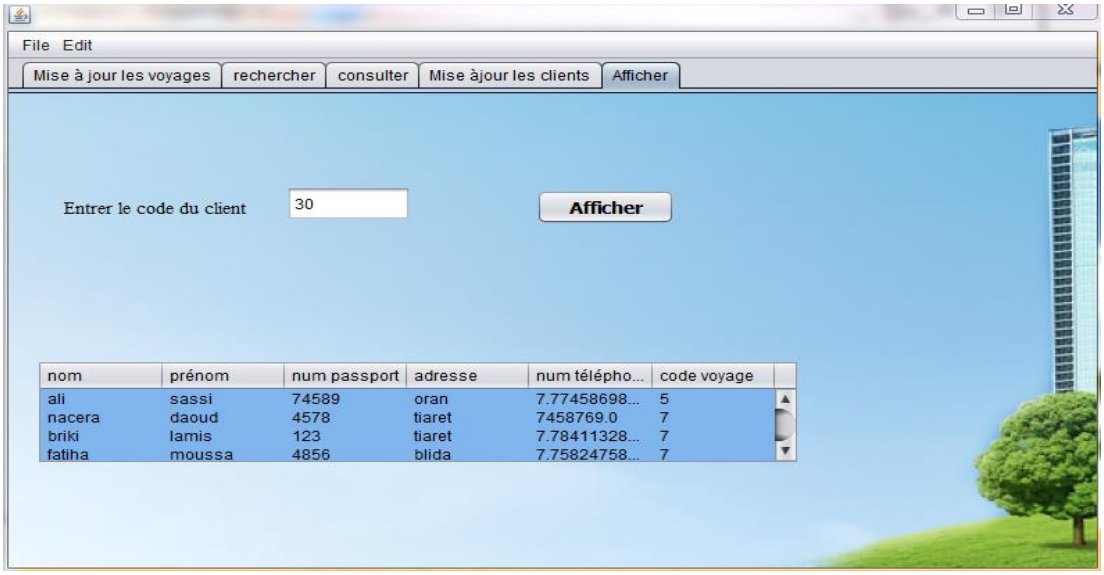

Figure 48 : Affichage des clients

Conclusion :

Dans ce chapitre, nous avons décrit brièvement le processus de réalisation de notre application en spécifiant l'environnement de développement, l'implémentation de la base des données et la démarche suivie pour la réalisation. En effet, nous avons achevé l'implémentation et les tests de tous les cas d'utilisation, tout en respectant la conception

élaborée. En d'autres termes, nous détenons la version finale de l'application, installée dans notre environnement de développement.

Conclusion générale :

Notre projet consiste à réaliser une application de gestion de réservation de voyage répartie basée sur les composants EJB, Pour concevoir ce travail nous avons commencé par la de la base de données de l'agence touristique par l'utilisation de la méthode de développement du Système d'information (MERISE) , sa mise en œuvre avec le gestionnaire de bases de données MYSQL, puis le développement des composants EJB ,et enfin la concrétisation de l'application sous l'environnement de programmation java (NetBeans IDE).

Nous utilisons les composants EJB (Enterprise Java Beans), Puisqu'ils gardent les avantages de la programmation orientée-objet (l'encapsulation des données, la séparation entre les données, le traitement conception de données et la réutilisation du code), l'offre d'une meilleure persistance de données de la BD gérée, au-delà, des applications développées à base de ce modèle qui ne sont plus des blocs monolithiques mais le regroupement de "pièces" bien définies.

Nous avons utilisé principalement les EJBs qui s'exécutent dans un conteneur EJB, ce dernier fournit les services suivant : la transaction, la sécurité, la localisation et la persistance.

Ce projet a fait l'objet d'une expérience intéressante, qui nous a permis d'améliorer nos connaissances et nos compétences dans le domaine de la création des applications réparties.

En effet, ce travail n'est pas un modèle unique et parfait, c'est pourquoi nous restons ouverts à toutes les critiques et nous sommes prêts à recevoir toutes les suggestions et remarques tendant à améliorer d'avantage cette étude. Étant donné que tout travail informatique a été toujours l'œuvre d'une équipe.

Perspectives :

Comme futur travail, en préalable, il faut enrichir les connaissances de la technologie des Enterprise Java Beans en créant d'autres EJB suivant nos besoins pour le bon déroulement de l'application choisie. En revanche, nous devons terminer le développement de J2EE en utilisant un client léger qui peut utiliser notre application en utilisant un simple navigateur web, ceci en utilisant par exemple les JSP ou la technologie des services web.

Références bibliographiques :

I. Livres :

[1] Moussine Pouchkine «EJB 3. © Des concepts à l'écriture du code Guide du développeur» Dunod, Paris, 2006 ISBN 2 10 050623 4.

[4] David Durand, « Gestion de la Qualité de Service dans les Applications Réparties sur Bus Middleware Orientés Objet » laboratoire de recherche en informatique d'Amiens P18-21. 8/11/2006.

[6] Java 2 Enterprise Edition.

[10] Interopérabilité : Corba Constructions d'application réparties H. JONES.

[12] Cours "Systèmes et réseaux répartis" NFP 214 par Jacques LAFORGUE (jacques.laforgue@neuf.fr) version du 22/11/2009 slide numéro 46.

[16] CORBA : principes et mécanismes Développement d'applications distribuées (par Z. Mammeri – UPS)

[17] CORBA : des concepts à la pratique Jean-Marc Geib - Christophe Gransart - Philippe Merle Laboratoire d'Informatique Fondamentale de Lille Université des Sciences et Technologies de Lille Email : {geib, gransart, merle}@lifl.fr

[19] Denivaldo LOPES, CORBA

[27] Laboratoire SUPINFO des technologies Sun Préface d'Alexis Moussine-Pouchkine Des concepts à l'écriture du code Guide du développeur EJB3.

[31] EJB des concepts à l'écriture du code guide du développeur Laboratoire SUPINFO des technologies Sun Préface d'Alexis Moussine-Pouchkine

[33] **(**eithe Une introduction aux Enterprises Java Bean Michel RIVEILL – Université de Nice – Sophia Antipolis http://www.polytech.unice.fr/~riveill- riveill@unice.frm

[36] Moussine Pouchkine «EJB 3. © Des concepts à l'écriture du code Guide du développeur Dunod, Paris, 2006 ISBN 2 10 050623 4

[38] Jean- Patrick MATHERON, Comprendre Merise, Edition EYROLLES, 2005 ; P.45.

Il. Sites web :

[2]http://wapiti.telecomlille1.eu/commun/ens/peda/options/ST/RIO/pub/exposes/exposesser2010-ttnfa2011/lemarchand-hardy/architectures_p2p.htm.

[3] http://fr.wikipedia.org/wiki/Calcul_distribu%C3%A9.

[5] http://fr.wikipedia.org/wiki/Architecture_distribu%C3%A9e.

[8] heithemabbes (heithem.abbes@gmail.com), 2010/2011.

[9] http://bruce-eckel.developpez.com/livres/java/traduction/tij2/?chap=16&page=9

[11] http://schuler.developpez.com/articles/p2p/.

[13]http://www.commentcamarche.net/contents/1030-introduction-a-rmi-remote-method-invocation.

[15] http://www.trucsweb.com/aSP/trucs.asp?no=287&type=7(15Mai2013)

[20] http://wwwigm.univmlv.fr/~dr/XPOSE2006/jabbour/pages/framework/framework.html#intro

[21] http://www-igm.univ-mlv.fr/~dr/XPOSE2006/jabbour/pages/framework/framework.html

[22] http://fr.wikipedia.org/wiki/Microsoft_.NET

[23] http://fr.wikipedia.org/wiki/Framework_.NET

[24] http://www.siteduzero.com/informatique/tutoriels/developpement-c-net/avantages-denet-par-rapport-aux-autres-plateformes

[25] http://www.commentcamarche.net/contents/548-j2ee-java-2-enterprise-edition

[26] http://www.commentcamarche.net/contents/548-j2ee-java-2-enterprise-edition

[28] http://www.mistra.fr/tutoriel-jee-introduction/tutoriel-jee-composants.html

[29] http://www.jmdoudoux.fr/java/dej/chap-ejb.htm

[30] https://fr.wikipedia.org/wiki/Enterprise_JavaBeans

[32] http://www.jmdoudoux.fr/java/dej/chap-ejb.htm#ejb-1

[34] http://www.jmdoudoux.fr/java/dej/chap-ejb.htm#ejb-2

[35] http://www.jmdoudoux.fr/java/dej/chap-ejb.htm

[37] http://www.additeam.com/SSII/systeme-d%E2%80%99information-si/

[39]http://www.memoireonline.com/09/12/6100/m_Conception-et-realisation-d-une-application-de-suivi-de-distribution-des-medicaments-de-l-action-d.html

III. Mémoires :

[7] mémoire de fin d'étude, gestion de scolarité, promotion 2013-2014.

[14] Mémoire de fin d'étude, développement d'une application client-serveur avec java RMI, promotion 2012-2013.

[18] mémoire de fin d'étude, Développer une application repartie a base des Ejb, promotion 2014-2015