République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abou Bekr Belkaid - Tlemcen
Faculté des sciences de l'ingénieur
Département d'Informatique

MÉMOIRE DE FIN D'ÉTUDES

Pour l'obtention du diplôme d'ingénieur d'état en Informatique
Option: Informatique Industrielle

THEME:

Modélisation et implémentation d'une application Web e-commerce dans la plate-forme J2EE

Présenté par :
Mr BENMERZOUKA Habib Ridha
Mr MALIKI Fouad

Encadré par :
Mr BENAMAR Abdelkarim

Devant le jury:
Mr ABDERRAHIM M^ed El Amine Président
Mr BENADDA Belkacem Examinateur

Promotion 2007

Table des matières

Table des figures

Introduction Générale

Les systèmes informatiques actuels sont à la fois efficaces, fiables et complexes du fait qu'ils se basent sur des architectures puissantes et extensibles.Ces architectures ont évolué depuis l'apparition des premières architectures que sont les Mainframes ou encore les architectures un-tiers.Depuis, la notion d'architecture est devenu essentielle pour le développement des futures applications.

Bien que les applications sur Mainframes aient connu un succès remarquable, la tendance vers une nouvelle architecture s'est concrétisée par l'avènement du modèle Client/Serveur.

Ce type d'architecture est très répandu du fait qu'il s'est bien adapté aux applications sur des réseaux informatiques.On est arrivé à une époque où les besoins exigent que de nouvelles architectures logicielles prennent place pour améliorer les systèmes Client/Serveur classiques.Donc le défi s'est inscrit sur le développement des architectures distribuées.

Ce qui a favorisé l'émergence de ces architectures c'est l'apparition de nouvelles approches de développement telles que l'orienté objet, le paradigme des composants ou encore l'approche par aspect.La vision est devenue donc différente vis-à-vis de la conception des nouveaux systèmes logiciels du fait qu'on s'est orienté vers le développement de systèmes à base de composants.

Actuellement, trois technologies se partagent le marché industriel à savoir J2EE, CORBA et Dot Net pour fournir une infrastructure et une architecture permettant le développement de ces systèmes à base de composants.

La plateforme J2EE offre un standard de développement complet et riche de technologies afin de concevoir des applications extensibles, portables, fiables et robustes.

La norme CORBA de l'OMG s'attache plus spécifiquement à l'interopérabilité des systèmes hétérogènes.

La framework Dotnet de Microsoft s'est révélée l'architecture concurrente de J2EE.Elle offre une plateforme pour le développement d'applications fonctionnant sous les systèmes Windows mais développés dans différents langages.

Notre travail n'étant pas une étude comparative entre ces architectures, mais plutôt d'étudier une technologie particulière de la plateforme J2EE qui est les Enterprises

JavaBeans (EJB).

Ce travail est divisé en trois parties; dans la première partie, nous présentons les points nécessaires qui ont exigé le passage des architectures centralisées aux architectures distribuées ainsi que, l'évolution de ces dernières vers des architectures multi-niveaux sur lesquelles se base la plate-forme J2EE.

Dans le second chapitre de cette partie, nous ferons un tour d'horizon sur la notion de composants ; nous présentons les caractéristiques des composants, leur contribution dans le développement rapide des applications ainsi que les avantages qu'ils offrent par rapport à l'approche objet.

La deuxième partie étant plus importante du fait qu'elle contient le coeur du sujet, cette partie est abordée par une description générale de la plate-forme J2EE en citant les services et les composants qu'elle offre, ainsi que, la structure générale d'une application J2EE.

La deuxième section de cette partie est destinée aux EJB dans laquelle nous allons présenté les constituants d'un composant EJB (interface home, interface remote,...), les différents types de composants EJB et les étapes de développement d'un composant EJB, en se basant sur un exemple de bean session.

Cette partie est clôturée par la présentation détaillée des différents types de composant EJB, et cela, par la description de leur caractéristiques et leur cycle de vie.

Ce travail est conclu par une dernière partie qui est consacrée à la mise en oeuvre pratique des concepts théoriques des EJB dans une application e-commerce.Nous allons décrire dans cette partie les fonctions de base de notre application et le processus de développement suivi, de la spécification à l'implémentation.

Première partie

Etat de l'art des architectures distribuées

Chapitre 1

Evolution des architectures distribuées

1.1 Introduction

Les systèmes d'information des entreprises deviennent de plus en plus complexes et requièrent des systèmes informatiques fiables qui répondent aux besoins.

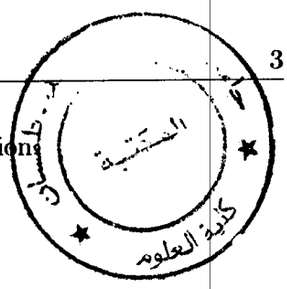
Dans ce contexte, la notion d'architecture logicielle applicative a favorisé le développement d'applications futures fiables et simples à maintenir, grâce aux progrès technologiques.

Pour en arriver à ce niveau de maturité, ces architectures ont évolué au fil du temps, allant de l'informatique centralisée des mainframes jusqu'aux nouvelles architectures N-tiers.

Dans ce but précis, ce chapitre introduira une description générale sur l'évolution des architectures distribuées.

Mais avant de présenter ces différentes architectures il s'avère important de présenter les trois niveaux d'abstraction d'une application :

- ***Le niveau de présentation*** : Il concerne tout ce qui est interaction entre l'application et l'utilisateur, Il gère les saisies au clavier, à la souris et la présentation des informations à l'écran.
- ***La logique applicative*** : Elle décrit l'ensemble des traitements à réaliser par l'application que l'on peut diviser en deux types :
  - *Traitements Locaux* : Ils regroupent le contrôle effectué au niveau du dialogue avec l'interface.
  - *Traitements globaux* : Appelés également logique métier, laquelle constitue l'application elle même.
- ***Le niveau de données*** : Il regroupe l'ensemble des mécanismes permettant



l'accès et la gestion des données stockées par l'applicatio

1.2 Architectures un-tiers

1.2.1 Présentation

Le concept d'architecture un-tiers consiste à regrouper les trois niveaux d'abstraction d'une application dans une même machine.On parle ainsi d'architecture centralisée.

Dans un contexte multi-utilisateurs, on peut rencontrer deux types d'architectures mettant en oeuvre une architecture un-tiers :

- Des applications sur site central (mainframe) ;
- Des applications réparties sur des machines indépendantes communiquant par partage de fichiers.

Les applications sur mainframe

L'évolution des applications fut marquée par l'apparition des mainframes qui proposaient un accès multi-utilisateurs.Dans ce contexte, les utilisateurs se connectent aux applications exécutées sur le serveur central (mainframe) à l'aide de terminaux passifs.En effet, l'intégralité des traitements, y compris l'affichage, qui est simplement déporté sur des terminaux passifs, sont pris en charge par le serveur central.Cette architecture est illustrée par la figure suivante :

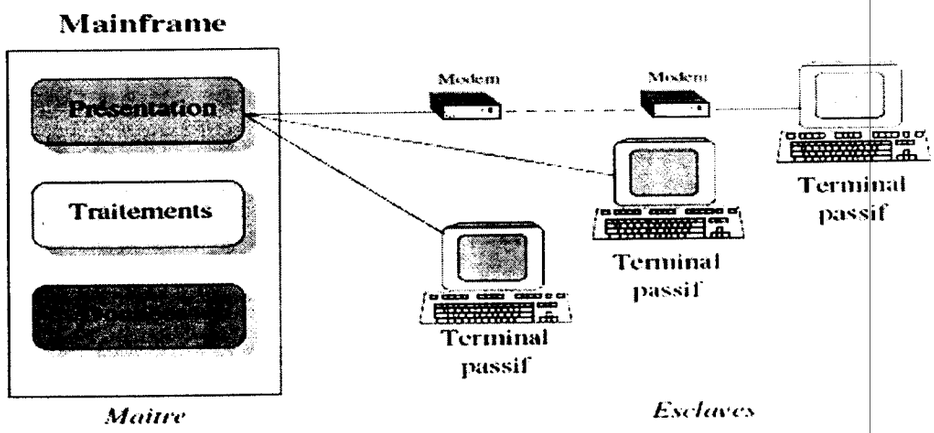


FIG. 1.1 – Architecture d'une application sur site central. [Lebl99]

L'émergence des interfaces graphiques de type Windows et Macintosh sur des micro-ordinateurs a démodé les applications sur mainframe qui utilisaient des interfaces en mode caractère.

Les applications un-tiers déployées

L'apparition des réseaux d'ordinateurs a permis le déploiement d'application un-tiers sur plusieurs machines autonomes.

Les utilisateurs se partagent des fichiers de données stockés sur un serveur commun, sachant que la gestion des données est prise en charge indépendamment par chaque poste.La gestion des conflits d'accès aux données doit être prise en charge par chaque programme de façon indépendante, ce qui n'est pas toujours évident [Lebl99].

1.2.2 Limitations

On constate que les applications sur site central souffrent d'une interface utilisateur en mode caractère et que l'exploitation des données communes dans les applications déployées sur des machines autonomes n'est pas fiable au delà d'un certain nombre d'utilisateurs [Lebl99].

Il a fallu donc trouver une solution basée sur les avantages des deux premières à savoir :

- une gestion centralisée des données (pour les applications sur site centrale) :
- l'interface utilisateur moderne (pour les application déployées sur des machines autonomes).

Ainsi est né le modèle client/serveur.

1.3 Architectures deux-tiers

1.3.1 Présentation du modèle de référence du Gartner group

Le Gartner group a défini un modèle de référence des différents types des clients/serveurs.

Ce modèle qui est illustré par la figure 1.2 définit la répartition des niveaux de l'application (présentation, traitement, données) entre le client et le serveur.

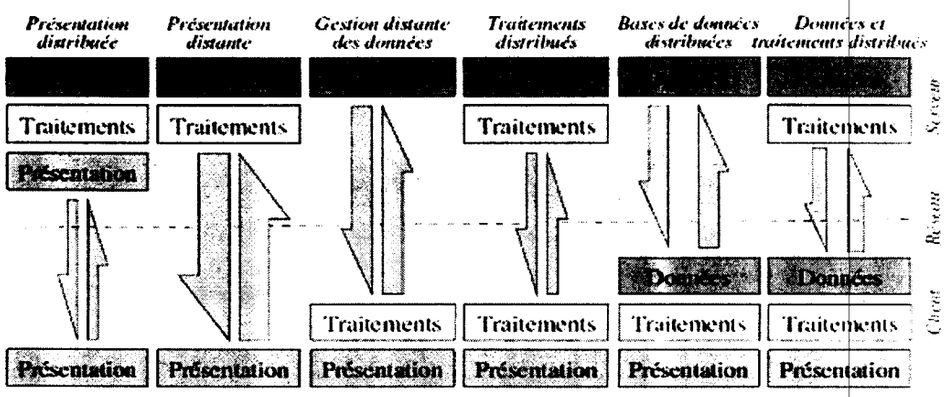


FIG. 1.2 – Les différents types de client serveur selon la classification du Gartner Group.

[Lebl99]

Le trait horizontal représente le réseau et les flèches entre le client et le serveur représente le trafic du réseau.

- **Présentation distribuée** : L'intégralité des traitements est prise en charge par le serveur, quant au client, il conserve une position d'esclave par rapport au serveur.Cette classification est jugée abusive.
- **Présentation distante** : Encore appelée client serveur de présentation, l'ensemble des traitements et données est géré par le serveur, le client ne prend en charge que l'affichage.Ce type d'application présente l'inconvénient d'engendrer un fort trafic du réseau.
- **Gestion distante des données** : Correspond au client/serveur de données est sans doute le modèle le plus répandu.L'application fonctionne dans sa totalité sur le poste client et la gestion des données est assurée par un Système de Gestion de Base de Données (SGBD) centralisé.
- **Traitements distribués** : Correspond au client/serveur de traitement.Les traitements sont distribués entre les clients et le serveur. Il s'appuie sur le mécanisme d'appel de procédures à distance (Remote Procedure Call).Cette architecture permet d'optimiser la répartition de la charge de traitement entre machines et limite le trafic du réseau.
  Ce modèle présente l'inconvénient de ce que les traitements doivent être connus du serveur à l'avance.
- **Bases de données réparties** : Il s'agit d'une variante du client/serveur de données dans laquelle une partie des données est prise en charge par le client :

ce modèle est intéressant si l'application doit gérer de gros volume de données.Si l'on souhaite disposer de temps d'accès très rapide sur certaines données ou pour répondre à de fortes contraintes de confidentialité.

- **Données et traitements distribués** : Ce modèle est très puissant et tire partie de la notion des composants réutilisables pour répartir au mieux la charge entre le client et le serveur.C'est bien entendu, l'architecture la plus difficile à mettre en oeuvre [Lebl99].

1.3.2 Présentation de l'architecture deux-tiers

Une architecture deux-tiers appelée aussi client/serveur de première génération ou client/serveur de données est représentée par le troisième modèle du Gartner group tel que le poste client se charge des traitements et de la présentation, quant à la gestion des données elle est prise en charge par le serveur.
Le dialogue entre le client et le serveur est illustré par la figure suivante :

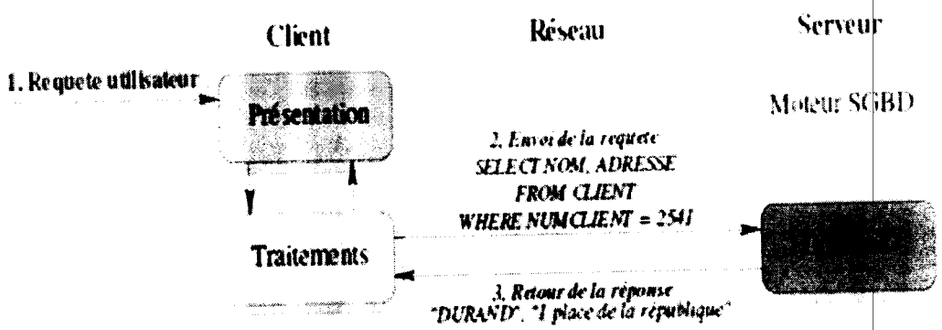


FIG. 1.3 – Accès aux données en mode deux-tiers.
[Lebl99]

Client : un programme qui envoie des requêtes au serveur.

Serveur : un programme qui traite les requêtes du client.

Pour permettre la communication entre le client et le serveur, il faut mettre en oeuvre des mécanismes relativement complexes qui sont en général pris en charge par un "middleware".

1.3.3 Notion de middleware

Définition

Un middleware appelé aussi **intergiciel** est l'ensemble des couches réseau et des services logiciels qui permettent le dialogue entre les différents composants d'une application répartie.

Objectif et rôle

L'objectif principal du middleware est d'unifier pour les applications l'accès et la manipulation de l'ensemble des services disponibles sur le réseau afin de rendre l'utilisation de ces derniers presque transparente [Lebl99].
La figure suivante présente la position du middleware entre le client et le serveur.

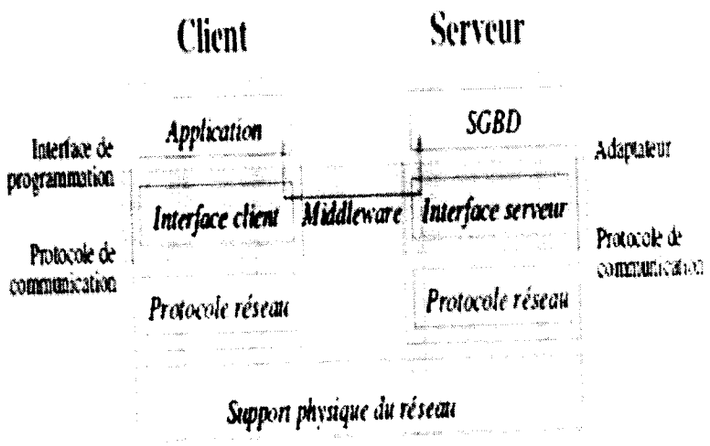


FIG. 1.4 – Positionnement du middleware.
[Lebl99]

Le middleware possède deux interfaces "interface client" et "interface serveur" qui s'intercalent respectivement entre l'application (cliente et serveur) et le protocole réseau.

- L'interface client propose une Interface de Programmation (API) qui fournit l'ensemble des fonctions permettant la communication avec le serveur ;
- L'interface serveur fournit un adaptateur qui rend les requêtes du client compréhensible par le serveur.

Services offerts par le middleware

- **Conversion :** Elle permet la communication entre des systèmes utilisant des formats de données différents ;
- **Adressage** : Elle permet la localisation du serveur distant sur le réseau ;
- **Sécurité** : Elle permet la confidentialité et l'intégrité des données en utilisant les mécanismes d'authentification et de cryptage ;
- **Communication** : Elle permet la transmission de messages entre le client et le serveur sans altération ; ce service doit gérer la connexion au serveur, la préparation de l'exécution des requêtes, la récupération des résultats et la déconnexion [Lebl99].

1.3.4 Avantages et inconvénients

Certes, le modèle client/serveur est très répandu, notamment, pour les applications de gestion exploitant des bases de données centralisées et gérées par un SGBD, tel que le poste client utilise une interface utilisateur riche GUI (Graphical User Interface) et se charge des traitements de l'application.Cette architecture présente toutefois des inconvénients :

- Le poste client est fortement sollicité ;
- Le client est lourd, parce qu'il supporte la grande majorité des traitements applicatifs ;
- La maintenance est très coûteuse et assez complexe.

Pour combler les lacunes de l'architecture deux-tiers, il est nécessaire de prendre en considération les points suivants :

- Rendre le client léger c'est-à-dire la gestion uniquement de la présentation ;
- Déporter les traitements sur un serveur intermédiaire entre le tiers client et le tiers serveur de données appelé serveur d'applications ;
- Minimiser les coûts de déploiement et de maintenance.

D'où l'orientation vers une architecture distribuée(trois-tiers et multi-tiers).

1.4 Les architectures trois-tiers

1.4.1 Objectif

L'objectif principal des architectures trois-tiers est de standardiser le développement des middlewares entre le client et le serveur.Dans ce but, l'architecture trois-tiers applique les principes suivants :

- Les données sont toujours gérées de façon centralisée ;

- La présentation est toujours prise en charge par le poste client ;
- La logique applicative est prise en charge par un serveur intermédiaire (serveur d'applications).

1.4.2 Historique des architectures trois-tiers

Les premières solutions de mise en oeuvre d'architectures trois-tiers proposaient un serveur d'applications centralisée exploité par les clients à l'aide du mécanisme RPC (Remote Procedure Call) propriétaire (chaque société développait son propre mécanisme RPC par exemple : UNIX, NetDDE, DCOM,...).Ces solutions étaient très complexes à mettre en oeuvre du fait du manque de standard.

Les serveurs de transaction

On a vu l'intérêt de décrire ce type de serveur puisqu'il constitue, à notre avis, une architecture trois-tiers du fait que les traitements des transactions sont centralisés et gérés par un serveur transactionnel, qui est interrogé par le client en utilisant une API unique tout en lui masquant la complexité de l'organisation des serveurs de données.

La révolution d'Internet

Ces dernières années, l'événement le plus marquant de l'informatique est bien évidemment celui d'Internet.Ce réseau mondial est rendu possible certainement par l'utilisation des standards reconnus tels que HTTP, HTML, TCP/IP, CGI et en se basant sur le modèle client/serveur :

- Le client est un navigateur web ;
- Un serveur web hébergeant les pages HTML ;
- Un serveur de données.

1.4.3 Présentation de l'architecture trois-tiers

C'est un modèle logique d'architecture applicatif qui vise à séparer très nettement les trois couches logicielles de l'application :

- La présentation des données qui est pris en charge par le client ;
- La logique métier (serveur d'application) ;
- L'accès aux données (serveur de données).

Cette architecture est représentée par la figure suivante :

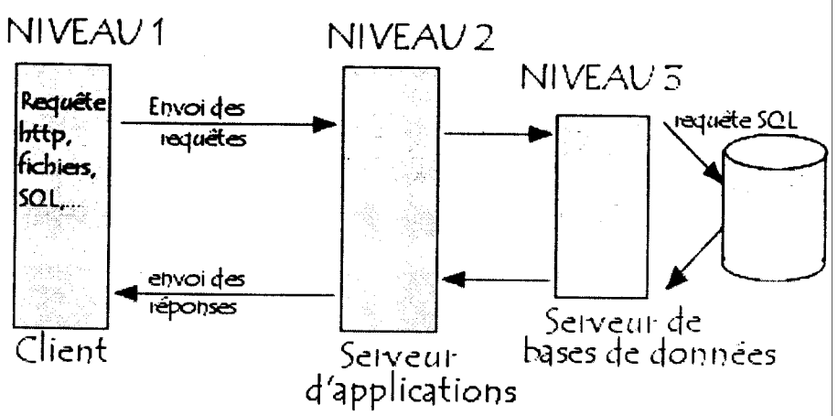


FIG. 1.5 – Architecture trois-tiers.
[CCM]

1.4.4 Avantages et inconvénients

Ce type d'architecture a réduit énormément la charge sur le client, en offrant une plus grande flexibilité des requêtes vers le serveur.Par contre, le serveur est fortement sollicité.

De plus, les solutions mises en oeuvre sont relativement complexes à maintenir et la gestion des sessions est compliquée [Lebl99].

Le juste équilibrage de la charge entre le client et le serveur semble atteint avec la génération suivante : les architectures N-tiers.

1.5 Les architectures N-tiers

Face aux lacunes des architectures précédentes, l'architecture N-tiers a été conçue pour développer des applications puissantes et simples à maintenir.

1.5.1 Définition

Une architecture N-tiers permet de distribuer plus librement la logique applicative, ce qui facilite la répartition de la charge entre tous les niveaux.

1.5.2 Caractéristiques

Une architecture N-tiers est caractérisée par :

- la mise en oeuvre d'une approche objet ;
- l'utilisation d'interface utilisateur riche ;
- la séparation des trois niveaux de l'application ;
- une grande capacité d'extension.

Ce type d'architecture est illustré par la figure suivante :

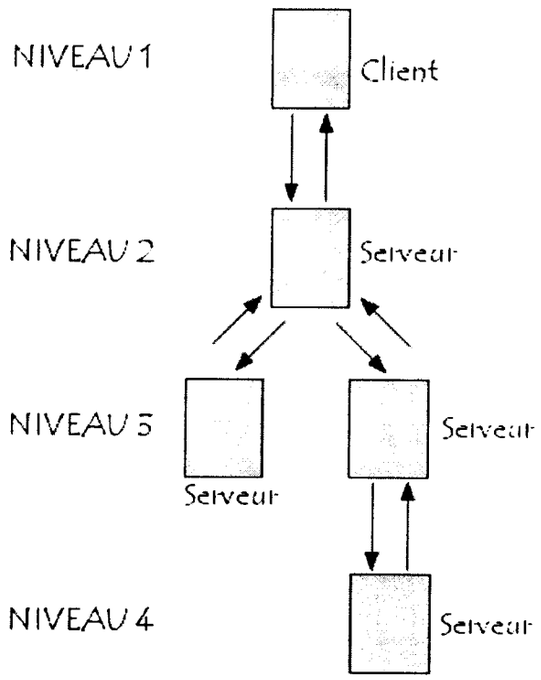


FIG. 1.6 – Architecture N-tiers.
[CCM]

Remarque : L'appellation N-tiers signifie la distribution de l'application entre de multiple services et non la multiplication des niveaux de service.

1.5.3 L'approche objet

Les concepts introduits par l'orienté objet, ont favorisé le développement d'applications N-tiers, et cela par l'utilisation de composants métiers réutilisables.

Il existe plusieurs technologies mettant en oeuvre ce type d'architecture dont on cite les plus répandus.

Les Java Beans

La spécification des Java Beans introduit par Sun Microsystem, définit un modèle de composants réutilisables, pour le développement d'application Java basé sur des composants côté client.Un Java Bean est caractérisé par :

- des propriétés qu'il expose à travers des accesseurs (getters/setters) ;
- des méthodes qu'ont peut invoquer comme tout objet Java ;
- des événements qu'il peut générer pour avertir d'autres composants.

Les Entreprises JavaBeans (EJB)

Les EJB est un standard pour le développement de composant Java côté serveur. Cette spécification définit un modèle de composants métiers réutilisables appelés Entreprises Beans qui sont déployés dans un serveur d'applications.

Ce modèle de composant qui fera l'objet de notre étude, sera détaillé dans les prochains chapitres.

Microsoft OLE COM ActiveX

Le modèle de communication COM permet de mettre en place une communication orientée objet entre des applications s'exécutant sur une même machine.DCOM est une extension de ce modèle pour les architectures distribuées.

Les contrôles ActiveX, anciennement dénommés OCX, sont des composants logiciels basés sur le modèle COM.Ils peuvent être intégrés à des applications où à des documents sous Windows [Lebl99].

1.5.4 La communication entre objets

Après avoir présenté les différents modèles de composants, il s'avère important de comprendre le principe de communication entre objets.

L'objectif est de rendre la communication entre objets presque transparente ; ainsi est né le concept de middleware objet.

Les différents middlewares objets sont :

1. ***Remote Method Invocation (RMI)*** : RMI permet à une application java, s'exécutant sur une machine virtuelle, d'invoquer les méthodes d'un objet hébergé par une machine distante.Pour cela, le client utilise une représentation locale de l'interface de l'objet serveur.Cette représentation locale est appelée stub et représente l'interface de l'objet serveur.
2. ***Le modèle CORBA*** : CORBA est l'acronyme de Common Object Request Broker Architecture qui est un standard de l'Object Managment Group (OMG).

L'objectif de ce consortium est de faire émerger des standards pour le développement d'applications distribuées hétérogènes à partir des technologies orientées objet.Cette norme définit un bus logiciel appelé Object Request Broker (ORB) qui permet l'interaction des objets CORBA entre le client et le serveur, il représente la partie centrale de l'architecture CORBA qui est illustrée par la figure 1.7.

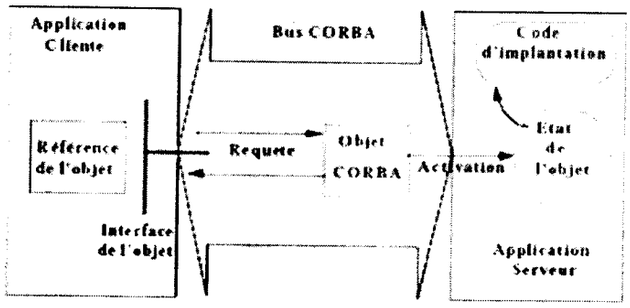


FIG. 1.7 – L'architecture du bus CORBA.
[GeGrMe99]

Chaque objet CORBA est développé dans un langage bien définit appelé Interface Definition Language (IDL) pour définir l'ensemble des services fournis aux applications clientes.

Ainsi, les problèmes liés à la localisation, le référencement et le nommage de ces objets sont totalement pris en charge par ce bus logiciel.

Dans ce contexte la norme CORBA définit aussi un modèle de composants réutilisables Corba Component Model (CCM) indépendant des systèmes d'exploitation et des langages de programmation.

1.6 Conclusion

Tout au long de ce chapitre, nous avons présenté les différentes architectures distribuées, on a commencé par présenter l'architecture un-tiers ensuite les limitations qui ont mené à une architecture plus performante qui est l'architecture deux-tiers où nous avons présenté la notion du middleware et les limitations de cette architecture.

On a aussi parlé de l'architecture trois-tiers et de ce qu'elle a apporté comme plus pour les architectures précédentes.Enfin, on a présenté l'architecture N-tiers ainsi que quelque modèles de composants qui ont permis à cette dernière de s'illustrer.

On constate que les architectures N-tiers offrent une vision cohérente du système et facilitent le développement d'applications ; mais cela n'empêche pas l'utilisation des autres architectures, selon les besoins.

Nous avons vu l'intérêt de consacrer le chapitre suivant à la notion de composant, qui se présente comme un concept fondamental pour le développement des applications multi-niveaux fiables et robustes.

Chapitre 2

Ingénierie des composants

2.1 Introduction

Il est indéniable que l'approche objet a beaucoup apporté au développement des systèmes logiciels, depuis son apparition, mais les besoins croissants des utilisateurs de systèmes 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 comme celles basées sur le paradigme de composants.

Certes, cette approche est basée sur des modèles à objets comme supports de spécification, de conception et d'implémentation, mais les différents aspects de développement logiciel sont différents de ceux introduits par l'orienté objet.

Ainsi, un composant peut être représenté par une classe ; les interfaces de composants peuvent être représentés par des interfaces de classes et les interactions entre composants peuvent être définies en termes d'associations et d'appels de méthodes [Ouss03].

Cependant, force est de constater que l'approche objet souffre d'un certain nombre de lacunes dont on cite les plus significatives :

- Faible niveau de réutilisation des objets dû au fort couplage des objets ;
- La structure des applications objets est peu lisible ;
- La plupart des mécanismes objets sont gérés manuellement (création des instances, gestion des dépendances entre classes, appels explicites de méthodes, etc.) ;
- Il y a peu ou pas d'outils pour déployer (installer) les exécutables sur les différents sites.

La notion de composant a été utilisée depuis longtemps dans la génie logiciel ; d'abord elle a désigné des fragements de code alors qu'aujourd'hui, elle englobe toute une unité de réutilisation.Cette approche était inspirée des composants de circuits

électroniques.

Aujourd'hui, le développement d'applications à base de composants constitue une voie prometteuse pour réduire la complexité de ce développement, le coût de maintenance et accroître le niveau de réutilisabilité.Deux principes fondamentaux doivent être respectés : acheter plutôt que construire et réutiliser plutôt qu'acheter [Ouss03].

Un des concepts fondamentaux des composants est la réutilisabilité qui a été introduite par McIlroy en 1968 dans une conférence de l'OTAN consacrée à la première crise du logiciel.

Partant de ce concept, on s'est ouvert une industrie de développement de logiciels à base de composants.

Ainsi, la construction des applications pourrait se faire plus rapidement en assemblant des composants préfabriqués.

2.2 Concept de composants

2.2.1 Notions et définitions

Il existe différentes définitions du concept de composant dont on cite ci-dessous les plus répandues :

Définition 1 *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.A software component can be deployed independently and is subject to composition by third parties.*

Cette définition a été donnée par Szyperski [Szyp98], elle décrit les caractéristiques d'un composant logiciel ainsi que ses dépendances avec son environnement externe.

Une autre définition d'un composant était donnée par Meyer [Meye99] :

Définition 2 *A software component is a program element with the following two properties :*

1. *It may be used by other program elements, or clients.*
2. *The clients and their authors do not need to be known to the component's authors.*

Enfin, Harris [HaHc99] a donnée aussi une bonne définition d'un composant :

Définition 3 *Un composant est un morceau de logiciel assez petit que l'on peut créer et maintenir, et assez grand pour que l'on puisse l'installer et en assurer le support.De plus, il est doté d'interfaces standards pour pouvoir interopérer.*

A partir des définitions précédentes, on déduit les propriétés suivantes, à savoir un composant est :

- *autodescriptif* : capable de disposer d'un mécanisme d'introspection permettant de connaître et de modifier dynamiquement ses caractéristiques ;

- *composable* : considéré comme une unité de composition, il est censé être connectable avec d'autre composants ;
- *configurable* : il doit être paramétrable via ses propriétés configurables selon un contexte d'exécution particulier ;
- *réutilisable* : il doit être une unité de réutilisation.Une étape d'adaptation sera sans doute nécessaire selon le contexte d'exécution ;
- *autonome* : il peut être déployé (c'est-à-dire diffusé et installé) et exécuté indépendamment des autres composants [Ouss03].

Il s'avère important de présenter les différents concepts pour la description des systèmes à base de composants :

- *Les composants* sont définis comme des unités de composition qui assurent une fonction bien précise ; ils peuvent être déployés indépendamment comme ils peuvent être composés avec d'autres composants ; ils possèdent ainsi un statut de réutilisation.Chaque composant possède une ou plusieurs interfaces pour interagir avec son environnement ;
- *Les interactions ou connecteurs* : il existe différentes formes d'interaction entre composants comme les appels de procédures, les événements,... qui sont des exemples d'interaction simple, quant aux interactions complexes comme les protocoles, les connexions avec des bases de données etc., ils sont représentés par des connecteurs ;
- *Les propriétés* : elles représentent les informations sémantiques des composants et leur interactions ;
- *Les contraintes/contrats* : représentent les moyens permettant à un modèle d'architecture de rester valide durant sa durée de vie et de prendre en compte l'évolution et le remplacement des composants logiciels dans cette architecture [Ouss03] ;
- *Une architecture* : elle définit un modèle pour décrire les composants et leurs interactions ;
- *La composition/assemblage* : permet de construire des applications complexes à partir de composants simples, la composition permet de lier un composant demandant des services à un composant offrant ces services.

2.2.2 Les trois dimensions d'un composant

Un composant peut être de deux natures :

- *Produit* : il s'agit d'une entité logicielle ou conceptuelle.Les composants logiciels et les bibliothèques de fonctions mathématiques en font partie ;

- *Processus* : un composant processus est un ensemble d'actions qu'il faut réutiliser pour obtenir un produit final.

Un composant produit ou processus doit refléter trois dimensions, à savoir :

- Son niveau d'abstraction ;
- Son mode d'expression ;
- Son domaine.

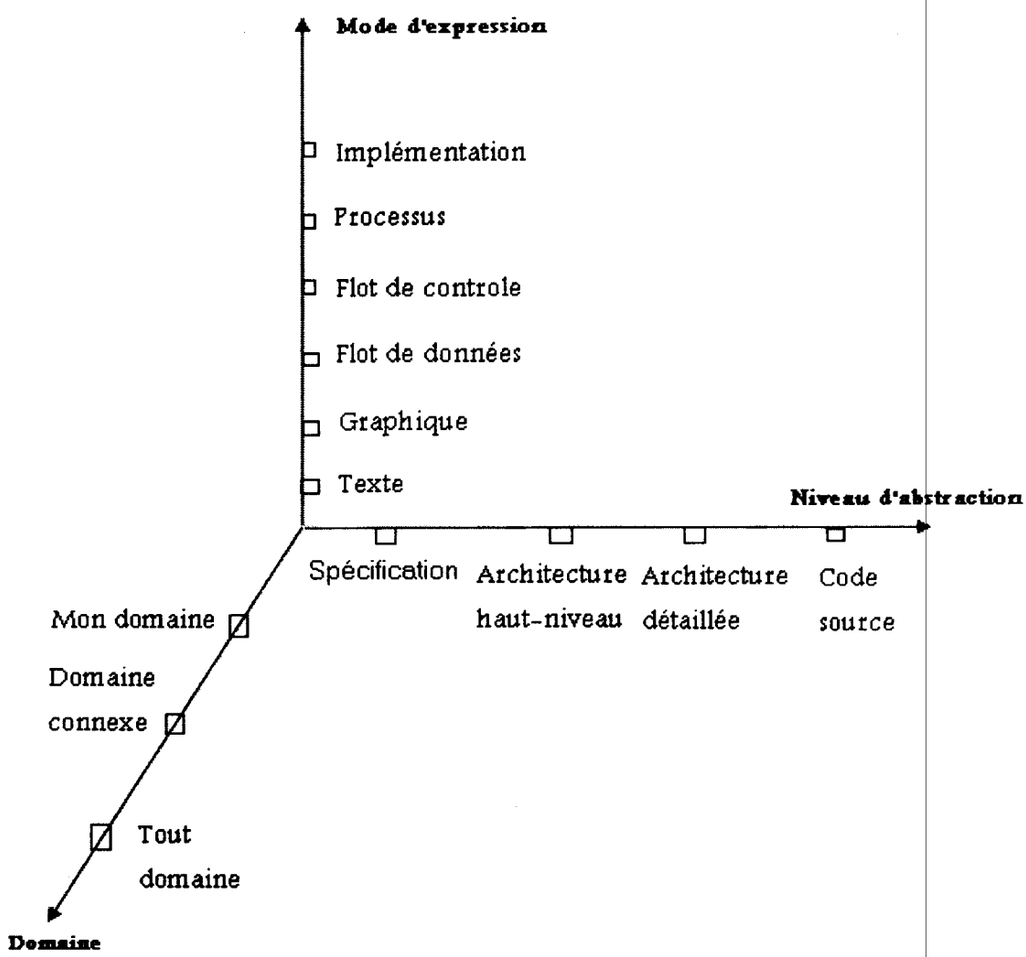


FIG. 2.1 – Les trois dimensions d'un composant.
[Ouss03]

La premiére dimension, permet d'exprimer les degrés de raffinement d'un composant de sa spécification jusqu'à son implémentation.

La deuxiéme dimension, permet de décrire les différents modéles de représentation d'un composant (modéle textuel, graphique,... etc).

Le mode d'expression d'un composant dépend de sa granularité qui recouvre aussi bien des unités atomiques(structures de données) que des unités complexes.

On peut distinguer trois niveaux de granularité :

- Les composants de fine granularité comme les classes, les fonctions ;
- Les composants de granularité moyenne comme les composants logiciels, les patrons de conception ;
- Les composants à forte granularité comme les frameworks.

la troisiéme dimension, décrit le degré de réutilisabilité du composant selon son appartenance à un domaine bien spécifique.Dans ce contexte on peut citer :

- *Les composants générique ou indépendant* : ce sont des composants qui ne dépendent pas d'un domaine particulier comme les structures de données ;
- *Les composants métiers ou dépendants du domaine* : ce sont des composants réutilisables à travers des applications du même domaine ;
- *Les composants orientés application* : ce sont des composants spécifiques à une application donnée, donc ils sont peu réutilisables.

2.2.3 Réutilisation d'un composant

L'idée de réutilisation d'un composant est simple du fait qu'on peut créer des applications à partir d'éléments existants.Ainsi, la réutilisation d'un composant permet de capitaliser sur l'existant en réutilisant un maximum de composants, de même qu'elle favorise l'accélération et la réduction du coût de développement de l'application.

2.3 Représentation d'un composant

Un composant est representé par :

- *Son type* : la définition abstraite du composant ;
- *Son implémentation* : elle englobe les aspects fonctionnels et les aspects non fonctionnels de son type ;
- *Son instance* : une instance de composant est une entité exécutable représentée par une référence unique ;
- *Interfaces* : une interface décrit l'ensemble des services fournis et requis, qui lui permet d'interagir avec son environnement, dont elle représente la seule partie visible.Elle est composée de :

1. **Points de connexion** : ce sont les points d'interaction entre le composant et son environnement.
2. **Services** : il y a deux types de services :

- *services fournis* : ils représentent le comportement fonctionnel du composant ;
- *services requis* : ils représentent les fonctionnalités dont il a besoin pour fonctionner.

2.4 Cycle de vie d'un composant :

Il regroupe l'ensemble des étapes de développement d'un composant, à savoir l'analyse, la conception, l'implémentation, le paquetage, l'assemblage et le déploiement.

- **Définition et analyse des besoins :** Cette étape permet de décrire les composants réutilisables en terme de fonctionnalités independamment de leur structure interne.
- **Conception des types de composants :** La conception consiste à spécifier les types de composants, à partir de l'étape précédente en utilisant un langage ; ce langage servira à spécifier les trois aspects d'un composant (interfaces, propriétés, mode d'interaction...).
- **Implémentation des types de composants :** Elle concerne l'implémentation de la partie fonctionnelle du composant, en utilisant un langage de programmation tout en respectant la spécification du type de composant.
- **Paquetage de composants :** Le paquetage permet de regrouper dans une même unité les interfaces du composant, son implémentation et des fichiers descripteurs qui aident à son déploiement.Par exemple, dans EJB et CCM, les paquetages sont respectivement dans le format JAR et ZIP.
- **Assemblage/Composition :** L'assemblage consiste à assembler tous les composants d'une application, dans une même unité, qui peut être à son tour empaquetée et réutilisée.
- **Déploiement :** le déploiement d'un composant consiste à l'installer dans son environnement.Ce processus se base sur le descripteur contenu dans le paquetage pour instancier le composant.
- **Exécution et utilisation :** Correspond à l'exploitation effective du composant, c'est-à-dire l'exploitation des services par les utilisateurs finaux.

Deuxième partie

Présentation des Entreprises JavaBeans (EJB)

Chapitre 3

L'architecture J2EE

3.1 Introduction

Le contexte de ces dernières années a favorisé l'émergence de nouvelles technologies, fortement liées à Internet, l'Intranet et à l'extranet.L'une d'entre elles s'est rapidement imposée en entreprise grâce à ses nombreux atouts et à ses performances.Cette technologie, spécifiée par Sun Microsystem avec l'aide de nombreux acteurs du monde informatique (Borland, IBM, Oracle, etc.) est connue sous le nom de Java 2 Enterprise Edition (J2EE).

J2EE présente une infrastructure d'accueil pour des composants développés dans le langage de programmation Java.

Ce chapitre couvre ainsi de façon détaillée les notions de base de la plate-forme J2EE.

3.2 Historique

L'émergence des nouvelles technologies que sont les systèmes distribués, sont à la base des technologies client/serveur.Historiquement, le premier mécanisme était celui d'appel de procédure à distance, qui consiste à appeler des fonctions d'un programme s'exécutant sur une autre machine ; ce mécanisme est connu sous le nom RPC introduit par les systèmes UNIX.

L'apparition des langages objets tels que Java a repris ce concept pour l'étendre à la notion de méthode d'un objet distant ; Java propose la technologie RMI pour implémenter ce mécanisme qui consiste à invoquer une méthode d'un objet s'exécutant dans un programme différent sur une autre machine.Toutefois, le développement de ce type d'application s'est révélé fastidieux et relativement complexe.L'inconvénient majeur résidait dans le fait qu'il était nécessaire de localiser l'objet avant d'invoquer

sa méthode.

Il est clair que les applications d'entreprise sont développées dans plusieurs langages du fait de la diversité des technologies; c'est donc de ce constat et afin de résoudre le problème de localisation d'objet qu'une nouvelle technologie a vu le jour : le bus CORBA, elle se base sur le protocole de communication IIOP (Internet Inter ORB Protocol) pour effectuer les appels de service d'objets distants.

Il est à noter que ces objets ne sont plus ici à entendre au sens des objets Java, car ils peuvent représenter une application à part entière.On parle d'objets CORBA qui sont spécifiés dans un langage IDL.

Cette technologie s'est révélée prometteuse, car elle répondait aux soucis de localisation, d'hétérogénéité et de communication, en standardisant les solutions à ces problèmes.Malgré ces avantages, l'inconvénient majeur était de développer des applications CORBA spécifiées en IDL et multi-langages, la charge de travail et la complexité technique des bus CORBA décourageaient rapidement les développeurs les plus tenaces [Durr03].

C'est pour répondre aux problèmes que posait cet ensemble de technologies qu'a été conçu J2EE.La technologie J2EE reprend les grandes idées et concepts vus précédemment, tels que la transparence de la localisation des objets a l'aide de JNDI (Java Naming and Directory Interface), des communications standardisées reposant sur RMI-IIOP... J2EE y adjoint une touche orientée métier (business) pour être au plus prés des préoccupations du client, une automatisation de la gestion des applications, des frameworks d'intégrations d'applications d'entreprise et un découpage selon le design pattern MVC (Model View Controller) pour obtenir ce que l'on appelle désormais des architectures multi-niveaux [Durr03].

3.3 Notions de base sur l'architecture J2EE

3.3.1 Organisation d'une architecture J2EE

La spécification J2EE a été conçue dans le but de fournir une plate-forme indépendante, portable, multi-utilisateurs, sécurisée et standard pour le développement d'application d'entreprise en Java.

J2EE définit un modèle d'architecture multi niveaux basé sur des composants, elle sépare très nettement les trois niveaux d'abstraction d'une application (présentation, logique métier, accès aux données).Cette architecture est illustrée par la figure suivante :

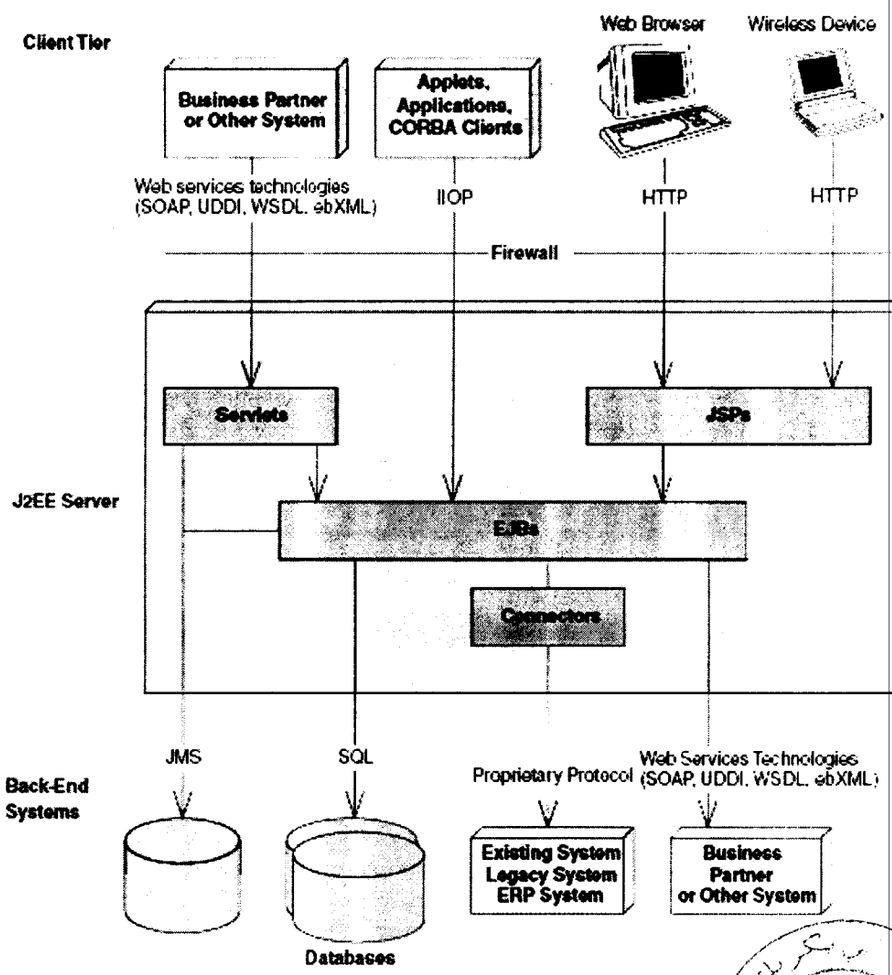


FIG. 3.1 – La plate-forme J2EE. [RoSrBr05]

A partir de la figure 3.1, on constate bien que cette architecture est organisée de la façon suivante :

1. **Tiers client :** Il représente l'application cliente (applet, web browser, application java).
2. **Tiers web :** Il représente le serveur web J2EE, il se charge de la partie présentation.

3. **Tiers métier :** Il représente le serveur métier J2EE (serveur EJB) pour implémenter la logique métier de l'application.
4. **Tiers données :** Il regroupe l'ensemble des données stockées dans le serveur de données (SGBD, LDAP,...).

3.3.2 Composants de l'architecture J2EE

- **La technologie EJB :** C'est un standard de développement de composants métiers, elle sera détaillée dans les prochaines sections.
- **Servlets :** Les servlets sont des programmes Java qui s'intègrent à un serveur d'application Web pour fournir le traitement coté serveur des requêtes issues d'un navigateur web client ; ils s'appuient sur un serveur web supportant la technologie Java serveur tel que Tomcat.
  Il est possible d'utiliser un programme écrit dans n'importe quel langage pour envoyer des requêtes à une servlet ; cela veut dire que le client peut être une simple page HTML, une applet Java ou un programme écrit dans un autre langage que Java.La servlet traite la requête et génère une sortie dynamique vers le client qui peut être une page HTML, XML ou encore un objet Java sérialisé.
- **Java Server Page(JSP) :** La technologie JSP est une extension des servelets qui offre un moyen simplifié pour écrire des servlets ; elle combine du code Java et du code HTML dans une page JSP.Toutefois, elle permet la gestion des pages web dynamique comme les ASP(Active Server Page), PHP(Personal Home Page), etc...

3.3.3 Les services de l'architecture J2EE

- **Remote Method Invocation-Internet Inter ORB Protocol(RMI-IIOP) :** C'est une extension de RMI, utilisée pour la communication entre les composants distribués.
- **Java Naming and Directory Interface(JNDI) :** C'est une API permettant l'accès aux services de nommage, elle peut être utilisée pour différentes raisons comme la localisation d'un composant EJB ou de n'importe quelle ressource, à travers le réseau.
- **Java DataBase Connectivity(JDBC) :** C'est une API permettant l'accès aux bases de données relationnelles.
- **Java Message Service(JMS) :** C'est une API qui permet la communication entre composants par envoi de message.

- **J2EE Connector Architecture(JCA) :** Elle offre un service d'accès à des informations d'entreprise existantes à partir d'application J2EE.
- **Java API for XML Parsing(JAXP) :** C'est une API qui permet le traitement des documents XML.
- **Java Transaction API/Java Transaction Services(JTA/JTS) :** C'est une API définissant des interfaces standards avec un gestionnaire de transactions.
- **JAVA-MAIL :** Elle permet d'envoyer des messages e-mails indépendamment du protocole.
- **JAVA IDL :** Permet l'implémentation de composants CORBA en java.
- **Java API for XML RPC(JAX-RPC) :** C'est la technologie principale pour le développement des web services, elle définit deux modèles, l'un basé sur les servlets et l'autre sur les EJB.
- **Java Authentication and Authorization Service(JAAS) :** Elle prend en charge l'aspect sécurité des opérations.
- **Java Management Extension(JMX) :** Elle fournit des extensions permettant de développer des applications web de supervision d'applications.

3.3.4 Les architectures web logicielles

Pour réaliser des applications web, on fait appel aux architectures logicielles qui offrent l'ensemble de logiciels permettant de rendre ces applications très performantes.On distingue deux types d'architectures web :

1. **Les architectures web statiques :** Ces architectures sont les premières mise en oeuvre ; elles permettaient de présenter des informations personnelles ou professionnelles sous une forme figée dans une page HTML ; elles concernent les sites web classiques.Pour réaliser une application web statique, il faut utiliser un serveur web pour héberger les différentes pages HTML sollicitées par l'internaute. L'inconvénient que présentent ces architectures est que les données contenues dans les pages HTML, ne sont pas issue d'une base de données, c'est-à-dire que la mise à jours des données est faite manuellement par l'administrateur du site, ce qui rend difficile sa maintenance.
2. **Les architectures web dynamiques :** Contrairement aux architectures web statiques, les architectures web dynamiques proposent un contenu dynamique dont les données sont généralement issues d'une base de données ; aussi elles peuvent récupérer des informations à partir des applicatifs des systèmes d'information de l'entreprise, dans le cas des applications web intranet telles que les ERP(Enterprise Resource Planning), les mainframes, les SGBD, etc.... Elles reposent sur un serveur web classique permettant de récupérer les données issues

d'un système de stockage.

Le système en question peut être basé sur des scripts CGI, des servlets, php, serveurs d'applications J2EE, PERL, etc....

Généralement, le serveur web et les serveurs d'applications J2EE se trouvent sur le même serveur physique, mais cela n'est pas obligatoire car ils peuvent être répartis sur différents serveurs physiques s'inscrivant dans le cadre d'architecture multi-niveaux.

3.3.5 Structure d'une application J2EE

Une application J2EE doit répondre à n'importe quel type de client que ça soit une simple page HTML, une applet, une application java s'exécutant sur une machine de bureau ou même un client java sur un périphérique portable comme un Assistant Numérique Personnel (ANP)ou un téléphone cellulaire.Un serveur d'application J2EE est basé sur la notion de conteneur web et de conteneur EJB.

Le conteneur web exécute des applications web qui prennent la forme d'une archive contenant des pages jsp ou des servlets s'exécutant sur un serveur web [RoSrBr05].

Un conteneur EJB fournit l'environnement d'exécution pour les entreprises java beans ainsi que la gestion de leur cycle de vie.

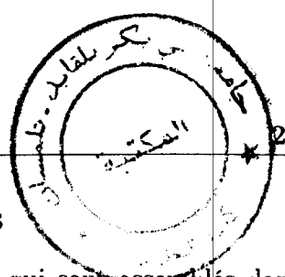
Un serveur web et un conteneur EJB fournissent des services aux EJB.

3.3.6 Structure d'une application web J2EE

Une application web J2EE est aussi nommée webapp dans le jargon J2EE : elle représente une archive web qui se présente sous la forme d'un fichier WAR et est structurée de la façon suivante :

- L'ensemble des pages JSP, HTML et images constituant le site.
- Un répertoire WEB-INF qui contient les éléments nécessaires au bon fonctionnement du site, à savoir :
  - *Les frameworks, tels que Struts, qui sont stockées dans le sous- répertoire lib ;*
  - *Les bibliothèques tel que Log4j, qui sont stockées également dans le sous-répertoire lib ;*
  - *Les servlets, sous la forme de classe java, qui sont stockés dans le sous-répertoire classes ;*
  - *Les composants javaBeans et les classes java standards, par exemple les applets qui sont également stockés dans le sous-répertoire classes.*

Elle contient aussi un descripteur de déploiement au format xml, utilisé par le serveur web.



3.3.7 Structure des Entreprises Java Beans

Les EJB contiennent la logique métier de l'application qui sont assemblés dans une archive (.jar) qui contient un ou plusieurs composants EJB.

Cette archive regroupe les classes java des EJB, et un descripteur de déploiement au format xml, comme dans le cas de la webapp.Ce descripteur est destiné au conteneur EJB du serveur d'application J2EE qui permet la configuration et la gestion efficace de chaque composant.

Le conteneur EJB se charge d'enregistrer chaque EJB dans l'annuaire JNDI, permettant ainsi aux clients de localiser le composant métier et d'y accéder [RoSrBr05].

3.3.8 Structure des applications d'entreprise

Les applications d'entreprise peuvent contenir plusieurs applications web et modules EJB ainsi que des bibliothèques externes nécessaire à l'exécution de l'application. Elle se présentent sous la forme d'une archive EAR (Entreprise ARchive) qui contient également un descripteur de déploiement application.xml qui détaille l'organisation des différents composants de l'application J2EE.

3.4 conclusion

Dans ce chapitre, on a établit la genèse de la technologie J2EE en passant par les technologies RPC, java RMI, et CORBA.On a souligné l'orientation des applications J2EE qui sont essentiellement destinées à mettre en place des architectures de type client léger, c'est a dire que le poste client ne dispose que d'un simple navigateur WEB pour accéder et interagir avec les systèmes logiciels.

Chapitre 4

Introduction aux Entreprises Java Beans

4.1 Introduction

Les systèmes à base de composants sont construits par assemblage de composants : il va de soi que tout modèle de composants vise à définir une infrastructure permettant la communication aisée entre ces composants, dans un environnement mutil-tiers.

C'est dans cette perspective que le modèle EJB s'inscrit pour fournir un modèle de composants standards distribués, portables, sécurisés, transactionnels et intéropérables.

Dans ce contexte ce chapitre donnera une vue compléte et claire de la spécification EJB 2.1.

4.2 Définition

Le modèle EJB est un standard pour le développement de composants côté serveur en java.Ce modèle de composant définit que n'importe quel composant peut être exécuté sur n'importe quel serveur d'application J2EE, les composants EJB appelés entreprise beans (beans) sont déployables et peuvent être importés et chargés dans un serveur d'application [RoSrBr05].

4.3 Caractéristiques des EJB

1. **Largement utilisés par les entreprises :** L'objectif principal des EJB est d'appliquer le principe " write once run anywhere " c'est-à-dire coder une seule

fois, exécuter n'importe où.Ainsi, les entreprises utilisant les EJB seront bénéficiaires du fait que leur utilisation est très répandue.Elles peuvent produire donc des logiciels de haute qualité.

2. **Portabilité :** Puisque la spécification des EJB est un standard, ils peuvent être utilisés sans être dépendants d'un fournisseur donné.
3. **Développement rapide d'application :** Le développement des applications est plus rapide avec les EJB, grâce aux services middleware offerts tels que les transactions, la sécurité et le polling.

Même si les EJB possèdent toutes ces vertus, il est important de savoir quand est ce qu'on les utilise.Voici quelques raisons pour lesquelles l'utilisation des EJB est vivement recommandée :

- Besoin de remoting (accès à distance).
- Utilisation des transactions distribuées.
- La sécurité.
- La persistance.
- Intégration du système réalisé avec des systèmes existants.
- La scalabilité c'est-à-dire l'adaptabilité à la charge des clients....

4.4 Les rôles

La spécification des EJB définit six rôles qui sont utilisés pour réaliser les tâches de développement et de déploiement ainsi que la définition du déploiement des composants du système :

1. **Le fournisseur d'entreprise bean :** C'est le développeur qui est responsable de la création des composants EJB.Ces composants sont regroupés dans un fichier jar spécial [Ecke00].
2. **L'assembleur d'application :** C'est l'architecte de l'application ; son rôle est de construire une application à partir d'un ensemble de composants.L'assembleur d'application est le client des entreprise beans fourni par le fournisseur d'entreprise bean.
3. **Le déployeur :** Après que l'application eut été assemblée, elle doit être déployée sur un ou plusieurs serveurs d'applications. Le déployeur prend en considération :
   (a) La sécurité du déploiement par l'utilisation d'un parfeu et d'autres mesures de protection.
   (b) Le choix du matériel qui fournit le niveau requis de qualité de service.

(c) L'utilisation de matériel et d'autres ressources de manière redondante pour la fiabilité et la tolérance aux pannes.

4. **L'administrateur système :** C'est le rôle le plus important de l'ensemble du système : mettre en place et faire fonctionner.La gestion d'une application distribuée consiste à ce que les composants et les services soient tous configurés et interagissent ensemble correctement [Ecke00].
5. **Le fournisseur du conteneur et du serveur EJB :** Fournit un environnement d'exécution et les outils qui sont utilisés pour déployer, administrer et faire fonctionner les composants EJB [Ecke00].
6. **Vendeur d'outil :** Pour faciliter le développement de composants, il existe plusieurs IDE (Integrated Developement Environment) permettant de construire, contrôler et maintenir les composants.Les IDE les plus populaire sont JBuilder, NetBeans et Eclipse.La plupart de ces outils permettent la modélisation des composants avec UML (Unified Modeling Language).

La figure 4.1 illustre les différent rôles des EJB.

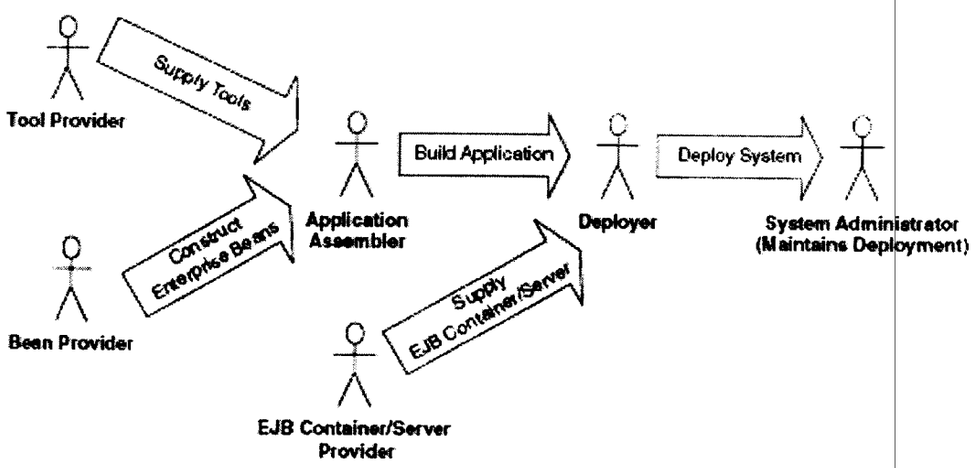


FIG. 4.1 – Les rôles des EJB.
[RoSrBr05]

4.5 Les concepts fondamentaux des EJB

Après avoir introduit la motivation derrière les EJB, ainsi que les différents rôles, nous les étudierons plus profondément dans ce qui suit :

L'objectif est de comprendre les types des entreprises beans, comment est composé un entreprise bean incluant la classe bean, l'interface home, l'interface locale home, l'interface remote, l'interface locale, le descripteur de déploiement et le fichier ejb-jar.

4.6 Définition d'un entreprise bean

Un entreprise bean est un composant logiciel côté serveur, qui peut être déployé dans un environnement distribué multi tiers, il peut être composé d'un ou de plusieurs objets java ; les clients font appel à un entreprise bean à travers une interface.Cette interface ainsi que l'entreprise bean doivent être conformes à la spécification EJB.

La spécification exige que l'entreprise bean expose certaines méthodes qui permettent au conteneur de gérer le cycle de vie des différents entreprises beans.

Notez bien qu'un client d'un entreprise bean peut être une servlet, une applet ou un autre entreprise bean.Dans le cas d'un entreprise bean, la requête du client vers le bean, peut se traduire par des appels à d'autres bean.Cette idée est très puissante du fait qu'elle permet de subdiviser une tâche complexe d'un bean en lui permettant de faire appel à d'autres bean effectuant les sous tâches.

4.7 Les types des beans

La spécification EJB 2.1 définit 3 types de composants :

1. **Les beans sessions** : Ils sont utilisés pour représenter des cas d'utilisation ou des traitements spécifiques du client ; ils représentent les opérations sur les données persistantes comme l'accès à une base de données, l'appel d'un système légataire ou l'addition de nombres, mais non les données persistantes elles mêmes.Il existe deux types de bean sessions : sans état (stateless) et avec état (statefull).
2. **Les beans entités** : Ils modélisent les données persistantes, ils peuvent être un produit, une commande ou un employé etc..., Ils peuvent être partagés par plusieurs clients comme les données d'une base de données.
3. **Les beans orientés messages** : Ils sont similaires aux beans sessions, c'est-à-dire ils réalisent des traitements.L'unique différence est qu'ils peuvent être appelés seulement par l'envoi d'un message, par exemple, un bean peut recevoir un message indiquant l'état du stock.

Sun a fournit ces différents types de composants pour répondre aux besoins des entreprises, la figure4.2 illustre les différentes possibilités d'interactions entre clients et composants EJB :

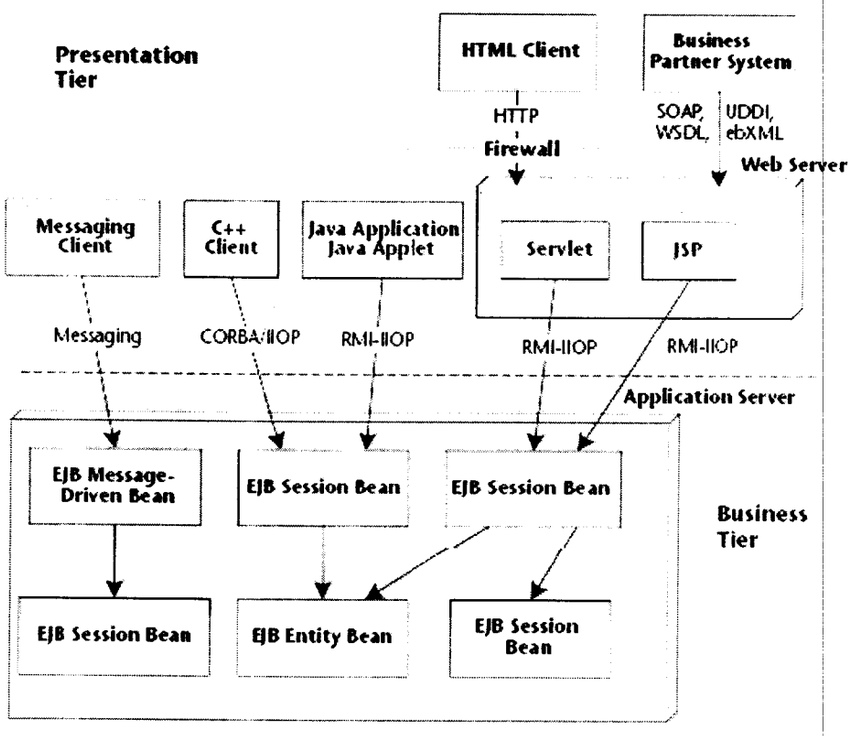


FIG. 4.2 – Les intéractions des clients avec les composants EJB. [RoSrBr05]

4.8 Notion d'objets distribués

La notion d'objet distribué est fondamentale afin de comprendre le principe du système de communication des EJB.Nous avons vu donc l'intérêt de présenter les concepts de base des objets distribués.Ce principe est illustré par la figure 4.3.

Considérons un programme qui désire invoquer une ou plusieurs méthodes qui se trouvent sur une machine distante.L'idée est de rendre l'appel de méthodes transparent au client comme si l'objet était sur la machine locale.

L'appel de méthode du client est dirigé vers un objet local appelé "Stub".Ce dernier masque la communication réseau.En se basant sur le principe des Sockets, le stub appel un objet distant coté serveur appelé Skeleton qui masque aussi la communication réseau à l'objet distant.Ce dernier délègue l'appel de méthodes à l'objet distant pour réaliser le traitement approprié.Le résultat retourné par l'objet est envoyé au skeleton qui à son tour l'envoi au stub.Enfin, le stub retourne effectivement le résultat au client.

Quand un client invoque une méthode, il a l'impression qu'il appelle l'objet direc-

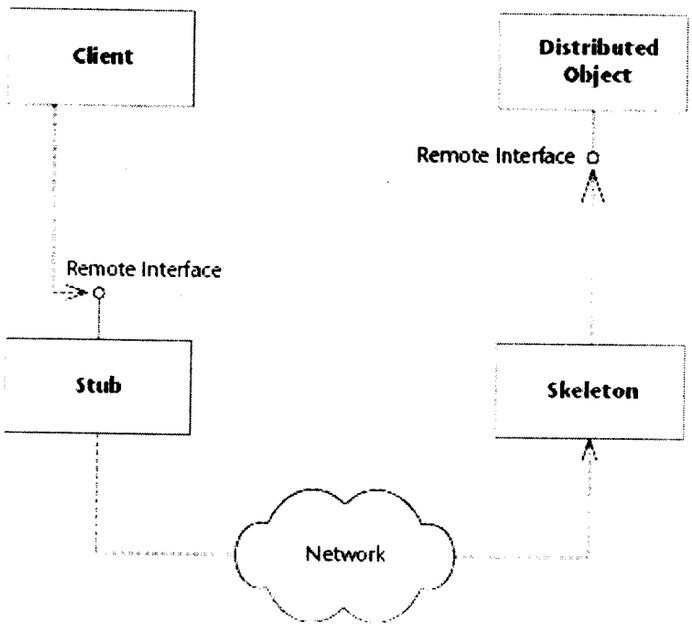


FIG. 4.3 – Les objets distribués.
[RoSrBr05]

tement, mais réellement, il fait appel à un stub qui implémente l'interface de l'objet distant et clone ses méthodes.

On peut utiliser plusieurs technologies pour l'implémentation de l'objet distant comme : Java RMI-IIOP (SUN), DCOM (Microsoft) et CORBA (OMG).

4.9 Les constituants d'un composant EJB

Un composant EJB est composé d'un ensemble de fichiers qui sont :

- **La classe Bean :** C'est une classe Java qui contient l'implémentation des traitements réalisés par le bean ; elle est soumise à certaines règles nécessaires au bean pour qu'il puisse être exécuté dans n'importe quel conteneur EJB.Ces règles diffèrent d'un type d'EJB à un autre :
  - *bean session* : La classe bean contient une logique liée à un traitement.
  - *bean entité* : La classe bean contient une logique liée à des données.
  - *message driven bean* : La classe bean contient une logique orientée message.

  La spécification EJB définit quelques interfaces standards qui peuvent être implémentées par la classe bean lui permettant d'importer certaines méthodes

utilisées par le conteneur EJB pour la gestion des beans, telles que l'interface javax.ejb.EnterpriseBean

- **EJB Object :** L'invocation de méthodes par un client ne se fait pas directement vers le bean.Il existe une couche intermédiaire entre le client et le bean. qui est l'EJB Object.Quand un client appelle une méthode, cet appel est intercepté par l'EJB Object ; ce dernier fait appel aux API middlewares nécessaires au fonctionnement tels que les transactions, la persistance, la sécurité ,....Les appels interceptés sont délégués au bean pour effectuer les traitements demandés.De même que le résultat est retourné aux clients à travers l'EJB Object.
  La figure suivante illustre ce principe :

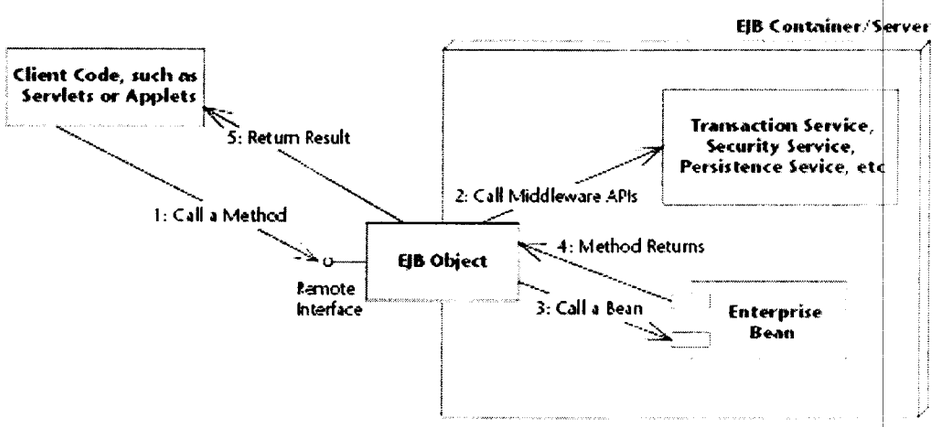


FIG. 4.4 – EJB Object.
[RoSrBr05]

- **Interface Remote :** Elle contient l'ensemble des méthodes métiers qui sont appelées par le client.Elle doit respecter certaines règles définies par la spécification EJB.Elle permet aussi à l'EJB Object de connaître les différentes méthodes de la logique métier.
- **Home Object** : Un client ne peut instancier L'EJB Object car il peut se trouver sur une machine différente de celle du client (c'est le cas des applications distribuées).Donc, la question qu'on peut se poser est comment acquérir une référence de L'EJB Object pour effectuer les appels de méthodes ? La réponse est que la spécification EJB 2.1 exige l'utilisation d'un autre objet appelé Home Object.
  Un Home Object a pour rôle :
  - La création d'un EJB Object.
  - La suppression d'un EJB Object.

- La recherche des EJB Objects existants.

Notez bien que le Home Object est spécifique à chaque conteneur EJB tout comme l'EJB Object.

- **Interface Home :** Puisque le Home Object est le constructeur (en anglais Factory) des EJB Objects, quelles sont les informations nécessaires pour l'initialisation de l'EJB Object ? Le conteneur a besoin de ces informations pour créer le Home object.Donc, ils sont fournis au conteneur par la spécification d'une interface Home.Cette interface définit des méthodes de création,de destruction et de recherche des EJB Objects.
- **Interface Locale :** Depuis la version 2.0 de la spécification EJB, on a introduit le concept d'objet local pour rendre l'appel au bean plus rapide et plus efficace.On peut utiliser cette méthode d'appel entre deux beans se trouvant dans le même conteneur.

  Un local object implémente une interface local tout comme l'EJB Object qui implémente l'interface remote.Généralement, l'appel d'un bean entité s'effectue à travers un bean session.
- **Descripteur de déploiement :** C'est un document XML qui contient plusieurs informations concernant les beans, à savoir le type des beans, les services demandés pour effectuer les différents traitements. le nom de la classe bean et des informations pour gérer le cycle de vie du bean ... .
- **Fichier spécifique au serveur d'application :** Chaque serveur d'application exige l'inclusion d'un fichier spécifique contenant des informations concernant le clustering, les noms JNDI, les sources de données,....
- **L'archive jar :** Après avoir généré les différentes classes beans, les interfaces locales, les interfaces remotes, le descripteur de déploiement et le fichier spécifique au conteneur, ils sont empaquetés dans un fichier est au format .jar.

la figure 4.5 montre les differents constituants d'une archive jar.

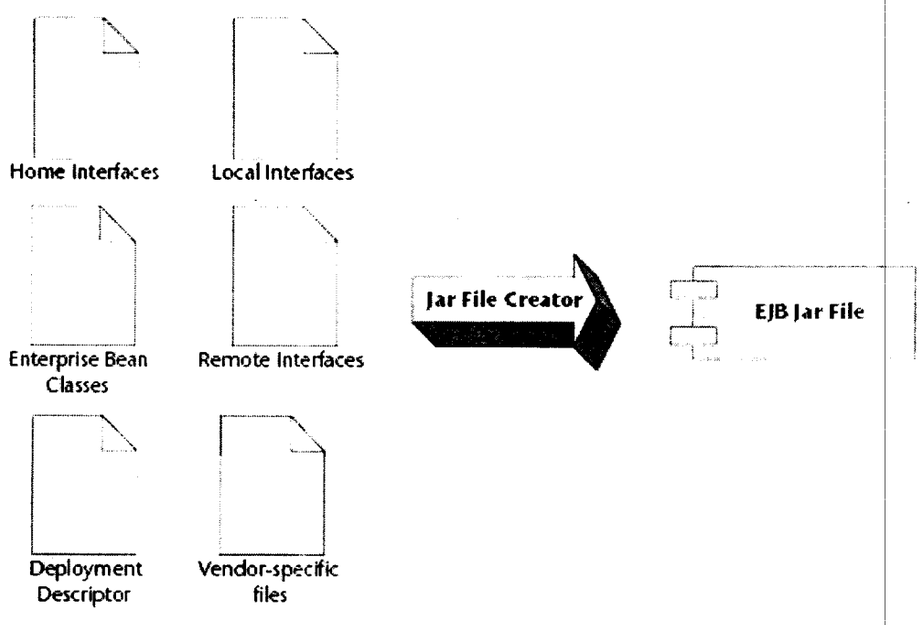


FIG. 4.5 – Création d'une archive jar.
[RoSrBr05]

Une fois ce fichier construit, il représente une unité déployable dans un serveur d'application.

Notez bien que cette archive peut contenir un ou plusieurs composants.

Remarque : Il existe plusieurs outils permettant la génération automatique de ce fichier tel que les IDE java

4.10 Les étapes de développement d'un EJB :

Pour réaliser un composant EJB, il faut suivre les étapes suivantes :

1. Ecrire les fichiers .java à savoir : la classe bean, l'interface remote, l'interface home, l'interface locale et n'importe quelle classe nécessaire au développement ;
2. Ecrire le descripteur de déploiement (ejb-jar.xml) ainsi que le fichier spécifique au conteneur ;
3. Compiler les fichiers java de l'étape 1(génération des .class).
4. Création de l'archive jar contenant le descripteur de déploiement, les fichiers .class et le fichier spécifique au serveur d'application ;
5. Déployer l'archive jar dans le serveur d'application ;

6. Configuer le serveur EJB ;
7. Démarrer le serveur EJB et confirmer le chargement du composant ;
8. Ecrire un programme client pour tester l'EJB.

Pour bien comprendre ces étapes, nous présentons l'exemple suivant qui consiste à développer un simple session bean stateless affichant un message HelloWorld :

1. **L'interface remote :** Lorsque vous créez l'interface remote, vous devez suivre les règles suivantes :
   - l'interface remote doit être publique ;
   - l'interface doit hériter de l'interface javax.ejb.EJBObject ;
   - chaque méthode de l'interface remote doit déclarer java.rmi.RemoteException dans sa section throws, en addition à des exceptions spécifiques à l'application ;
   - chaque objet passé en argument ou retourné par valeur (soit directement soit encapsulé dans un objet local) doit être un type de données valides pour RMI-IIOP.

   Voici l'interface remote de notre exemple :

```
package helloworld;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Hello extends EJBObject {
    public String hello() throws RemoteException;
}
```

2. **L'interface home :** Lorsque vous créez l'interface home vous devez suivre les règles suivantes :
   - l'interface home doit être publique ;
   - l'interface doit hériter de l'interface javax.ejb.EJBHome.
   - chaque méthode de l'interface home doit déclarer java.rmi.RemoteException dans sa section throws ;
   - la valeur retournée par une méthode de création doit être une interface remote ;
   - la valeur retournée par une méthode de recherche doit être une interface remote, java.util.Enumeration ou java.util.Collection (uniquement pour les beans entités) ;
   - chaque objet passé en argument ou retourné par valeur (soit directement soit encapsulé dans un objet local) doit être un type de données valides pour RMI-IIOP.

La convention standard de nommage des interfaces home consiste à prendre le nom de l'interface et à y ajouter à la fin Home.

Le code source de l'interface home est comme suit :

```
package helloworld;

import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import java.rmi.RemoteException;

public interface HelloHome extends EJBHome{
  public Hello create() throws CreateException,RemoteException;"
}
```

3. **La classe bean :** Maintenant que nous avons défini les interfaces de notre composant, on va implémenter la logique métier.

   Lorsque vous créez la classe d'implémentation de votre EJB vous devez suivre les règles suivantes :
   - la classe bean doit être publique ;
   - la classe doit implémenter une interface selon le type d'EJB (javax.ejb.SessionBean. javax.ejb.EntityBean. javax.ejb.MessageDrivenBean) ;
   - la classe doit définir les méthodes correspondantes aux méthodes de l'interface remote.Notez que la classe n'implémente pas l'interface remote ;
   - définir une ou plusieurs méthodes ejbCreate()qui initialisent votre EJB ;
   - chaque objet passé en argument ou retourné par valeur (soit directement soit encapsule dans un objet local) doit être un type de données valide pour RMI-IIOP.

   Notez que les méthodes ejbCreate() sont appelées par le conteneur EJB et sont utilisées pour contrôler l'état du bean.

   Le code source de notre classe bean est comme suit :

```
package helloworld;

import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;

public class HelloBean implements SessionBean {
  SessionContext sessionContext;
  public void ejbCreate() throws CreateException {
```

```
    }

    public void ejbRemove() {
    }

    public void ejbActivate() {
    }

    public void ejbPassivate() {
    }

    public void setSessionContext(SessionContext sessionContext) {
          this.sessionContext = sessionContext;
    }

    public String  hello() {
          return "Hello World";
    }
}
```

4. **Le descripteur de déploiement (ejb-jar.xml) :** Après avoir créé le composant, nous avons maintenant besoin de créer le descripteur de déploiement :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
  2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <display-name>EJB</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>Hello</ejb-name>
      <home>helloworld.HelloHome</home>
      <remote>helloworld.Hello</remote>
      <ejb-class>helloworld.HelloBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
```

```
    <assembly-descriptor>
      <container-transaction>
        <method>
          <ejb-name>Hello</ejb-name>
          <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
      </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Voici quelques explications sur les tags utilisés :
- <ejb-name> : le nom du bean.
- <home> : le nom de l'interface home.
- <local-home> : le nom de l'interface local home.
- <remote> : le nom de l'interface remote.
- <local> : le nom de l'interface local .
- <ejb-class> : le nom de la classe bean.
- <session-type> : le type de session bean (stateless ou stateful).
- <transaction-type> : transaction gérée par le conteneur ou par le bean.

Enfin, on écrit le fichier spécifique au conteneur.

Après compilation, les fichiers sont assemblés dans une archive .jar qui est déployée soit automatiquement soit par copie.Après le déploiement, le client a besoin d'appeler le bean, mais comment cet appel est-il effectué ?

5. **L'appel d'un Bean** : On va s'intéresser maintenant à la partie cliente.Un client (client du bean) peut être de deux types :
   (a) **Client Java RMI-IIOP :** C'est un client java qui utilise une interface JNDI pour localiser les objets à travers le réseau et une API JTA pour le contrôle des transactions.
   (b) **Client CORBA :** C'est le client qui appelle le composant EJB est écrit dans un autre langage que java comme C++ par exemple, les clients CORBA utilisent le COS Naming pour localiser les objets à travers le réseau.

   Le client doit suivre les étapes suivantes :
   (a) localiser le Home Object ;
   (b) utiliser le Home Object pour créer l'EJB Object ;
   (c) appeler les méthodes par l'EJB Object ;
   (d) détruire l'EJB Object.

4.11 La gestion des noms de composants dans JNDI

Tout type de composants, qu'il soit défini dans l'application J2EE ou qu'il soit externe, doit être nommé.Le serveur d'applications référence et maintient une liste de noms de ressources, au sein d'un annuaire, appelé JNDI (Java Naming and Directory Interface).

Lorsqu'une application J2EE souhaite accéder à une ressource, elle recherche son nom dans l'annuaire afin d'obtenir sa localisation.

JNDI est au coeur des applications J2EE ; sans cette fonctionnalité, il est impossible de connecter les briques de l'application entre elles, ou d'accéder à l'extérieur du serveur d'applications.L'annuaire est situé directement au niveau du serveur EJB.Ce dernier est le constituant du serveur d'applications, qui gère le ou les conteneurs EJB [Durr03].

4.11.1 Présentation de JNDI

L'annuaire JNDI fait partie intégrante des serveurs d'applications J2EE.Il sert à maintenir une liste de noms, qui correspondent à des composants J2EE ou à des ressources internes/externes au serveur d'applications.L'architecture JNDI est basée sur une API qui permet notamment aux clients de :

- rechercher des noms JNDI dans l'annuaire en vue de localiser des objets ou des ressources ;
- ajouter des références à la localisation d'objets dans l'annuaire et un nom qui permet de retrouver la référence ;
- supprimer les références, etc.

L'architecture JNDI est également basée sur une SPI (Service Provider Interface).Une SPI définit le moyen technique pour intégrer au framework JNDI, les systèmes de nommages et de répertoires existants.Cette architecture est illustrée par la figure suivante :

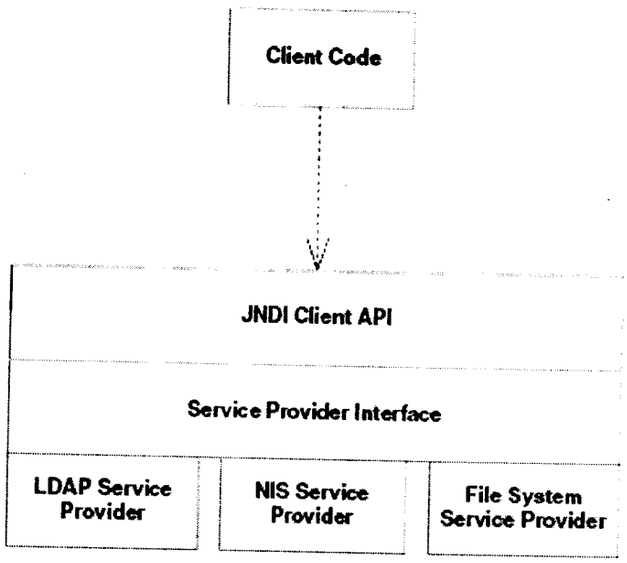


FIG. 4.6 – L'architecture de JNDI.
[RoSrBr05]

4.11.2 Localisation d'un composant EJB via l'annuaire JNDI :

Pour localiser un composant EJB, on doit associer un nom JNDI au Home Object, ce nom JNDI est spécifié dans le descripteur spécifique au conteneur.Lors du déploiement du composant, le conteneur fait le lien automatique entre le nom JNDI et le Home Object.Un client peut donc utiliser ce nom JNDI pour récupérer le Home Object en utilisant l'API JNDI.La figure suivante montre ce processus.

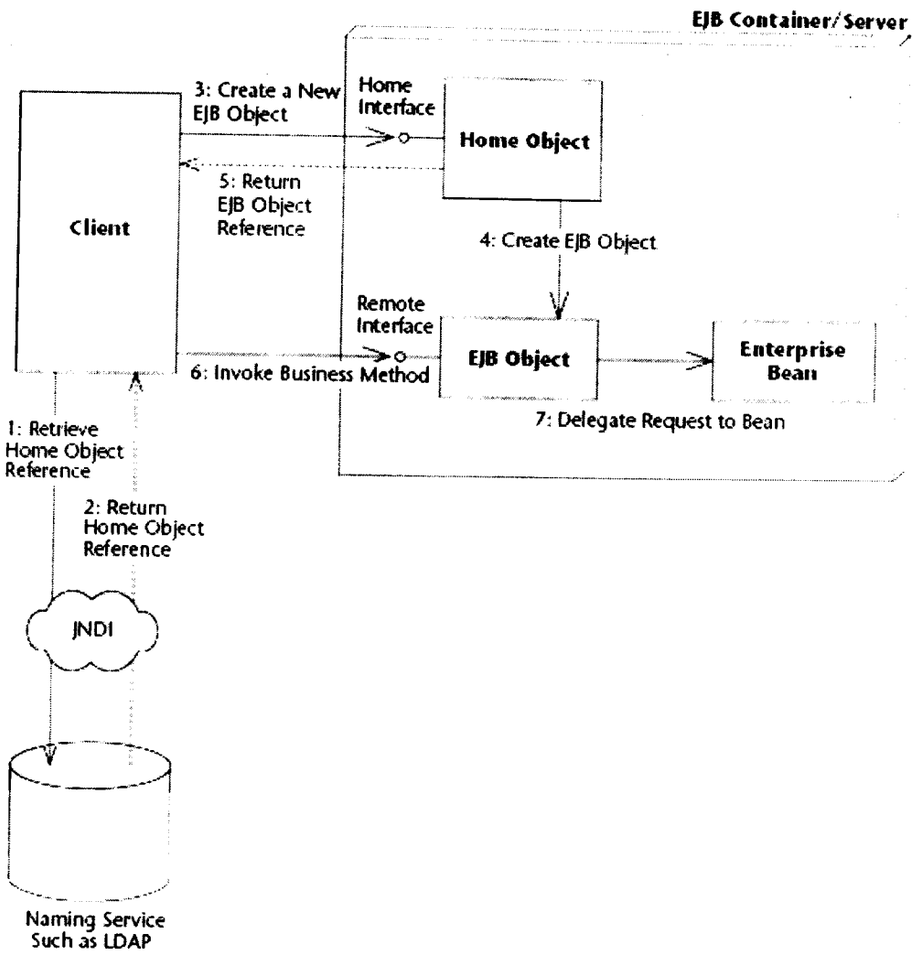


FIG. 4.7 – Acquisition d'un Home Object.
[RoSrBr05]

Après avoir expliqué le principe de JNDI, voici dans ce qui suit un code client de l'exemple précédent en prenant comme nom JNDI Hello.

```
package helloworld;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import javax.naming.InitialContext;
```

```
import java.util.Hashtable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
 public class HelloClient {
 public static void main(String[] args) {
   HelloHome helloHome = null;
   Hello helloRemote = null;
   Hashtable environment = new Hashtable();
   environment.put(Context.INITIAL_CONTEXT_FACTORY,
               "org.jnp.interfaces.NamingContextFactory");
   environment.put(Context.URL_PKG_PREFIXES,
               "org.jboss.naming:org.jnp.interfaces");
   environment.put(Context.PROVIDER_URL, "jnp://localhost:1099");
      // obtenir le contexte de nommage
   Context context = null;
   try {
       context = new InitialContext(environment);
     // Rechercher le nom JNDI
       Object ref = context.lookup("Hello");
     // recherche de nom jndi et transtypage en interface Home
      helloHome = (HelloHome) PortableRemoteObject.narrow(ref, HelloHome.class);
     // création d'un EJB Object
      helloRemote = helloHome.create();
      System.out.println(helloRemote.hello());
     } catch (NamingException ex) {}
       catch (RemoteException ex) {}
       catch (CreateException ex) {}
  }
}
```

4.12 Conclusion

Ce chapitre a définit les EJB et explicité les différents types d'EJB qu'il est possible de spécifier, ainsi que les notions de bases requises lors de la réalisation d'une application J2EE.

Nous constatons que le choix d'un tel modèle s'inscrit dans le développement d'application rapide (RAD).Cependant, les concepts invoqués dans ce chapitre requiert une certaine compétence particulière.

Chapitre 5

Classification des composants EJB

5.1 Introduction

Plusieurs types de composants sont définit dans la spécification EJB. ils doivent faire l'objet d'une étude approfondie afin de pouvoir opérer un choix portant sur un type de composant plutôt qu'un autre selon certains critères.

Dans ce contexte, ce chapitre se présente comme un complément du chapitre précédent, en couvrant les détails des différents types de composants.

5.2 Les bean sessions

5.2.1 Définition

Un bean session est un composant réutilisable qui implémente la logique métier : il diffère des autres composants par sa courte durée de vie qui est égale à la durée de la session ; il n'est pas partagé par plusieurs clients.

Selon le type de conversation entre le client et le bean, on distingue deux types de bean session :

1. **Bean session avec état(stateful) :** C'est un bean qui garde l'état entre l'invocation de méthodes c'est-à-dire si l'état du bean change durant l'appel de méthodes, cet état doit être disponible au client lors de la prochaine invocation.Par exemple, dans une application e-commerce, quand un client ajoute un produit dans un panier, l'état de ce dernier doit être sauvegardé.
2. **Bean session sans état(stateless) :** Ce sont les composants les plus simples à implémenter ; ils ne conservent aucun état conversationnel avec les clients c'est-à-dire qu'après chaque invocation de méthodes, le bean est soit détruit soit recréé par le conteneur. Puisque les beans stateless ne conservent pas d'état, toutes les

instances sont les mêmes pour tous les clients, c'est-à-dire que n'importe quel bean peut servir n'importe quel client.

La figure suivante illustre le principe d'invocation d'un session bean stateless :

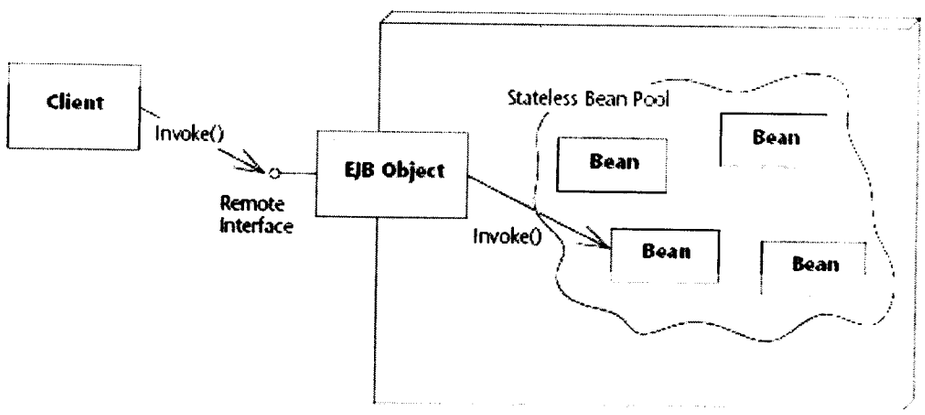


FIG. 5.1 – invocation d'un bean session stateless.
[RoSrBr05]

5.2.2 La gestion du pool des beans sessions

Il est nécessaire de connaître le fonctionnement interne du processus de gestion des instances du bean à l'intérieur du conteneur, pour bien maîtriser les différents concepts de chaque type des composants EJB ; ce processus diffère d'un type de composant à un autre.

La gestion du pool pour les beans sessions stateless est simple du fait que toutes les instances sont les mêmes aux clients, donc l'affectation d'un bean à un client se fait arbitrairement.

Par contre, pour les beans sessions statefull, la gestion du pool est plus compliquée, car l'état d'une conversation entre un client et un bean est sauvegardé, donc le conteneur ne peut pas affecter aléatoirement un bean à un client, pour résoudre ce probléme, le conteneur utilise la technique de changement de contexte utilisée dans les systémes d'exploitations.

Lorsque le systéme dispose d'une mémoire physique limitée et qu'il veut exécuter plusieurs applications, il utilise le disque dur comme mémoire virtuelle.

Supposons qu'on a deux applications A et B.L'application A est en exécution et l'application B veut s'exécuter, le systéme sauvegarde le contexte de l'application A

dans le disque dur, et par la suite, il charge le contexte de l'application B en mémoire pour qu'il puisse l'exécuter.

Le conteneur exploite cette technique pour limiter le nombre de bean dans le pool, pour ce faire, le conteneur sauvegarde l'état d'un bean dans le disque dur pour affecter ce bean à un autre client, cette technique est appelée passivation.

Quand le client de ce bean invoque de nouveau une méthode, l'état passivé est restauré vers un bean qui peut être différent de l'ancien bean ; cette technique est appelée activation.

- ***Passivation*** : Quand un client invoque une méthode d'un EJB Object et que tous les beans en pool sont associés à d'autre clients, le conteneur rend le bean le moins récemment utilisé (LRU) passif c'est-à-dire l'état du bean est stocké dans le disque dur en appelant la méthode ejbPassivate(), pour associer ce bean au client, ce qui est illustré par la figure suivantre :

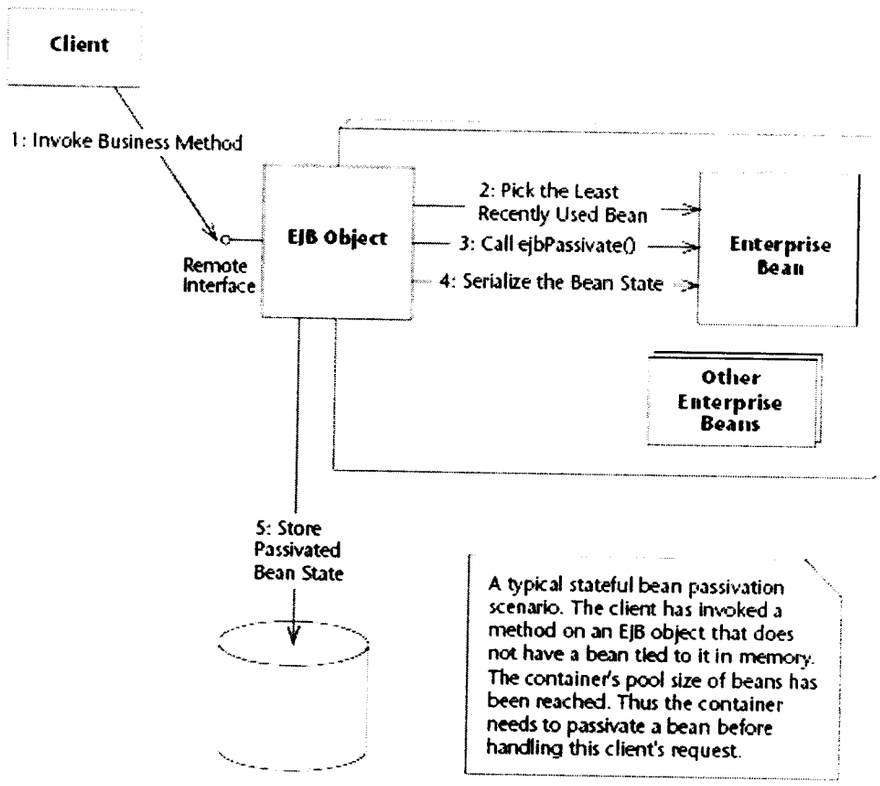


FIG. 5.2 – Passivation d'un session bean statefull [RoSrBr05]

- ***Activation*** : Si un client dont l'état est passivé, appelle une méthode d'un EJB Object, cette état est affecté a un bean, lequel est associé à ce client appelant

ainsi la méthode ejbActivate(), ce qui est illustré par la figure suivantre :

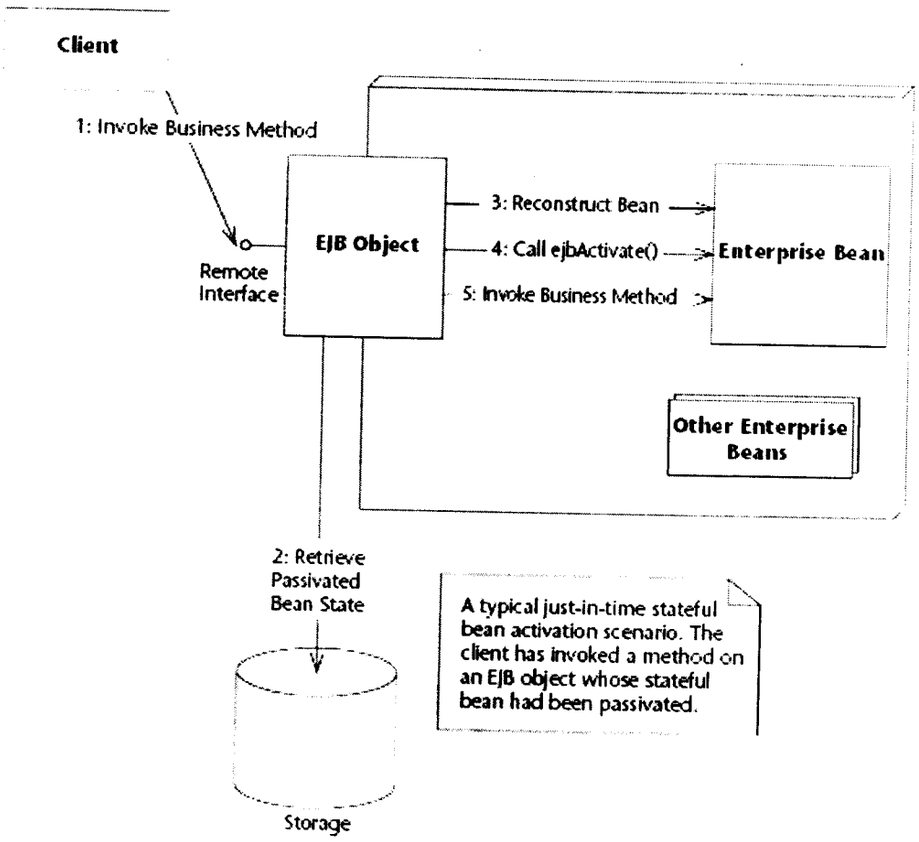


FIG. 5.3 – Activation d'un session bean statefull [RoSrBr05]

Le cycle de vie d'un bean session

Le cycle de vie d'un bean session stateless est illustré par la figure suivante :

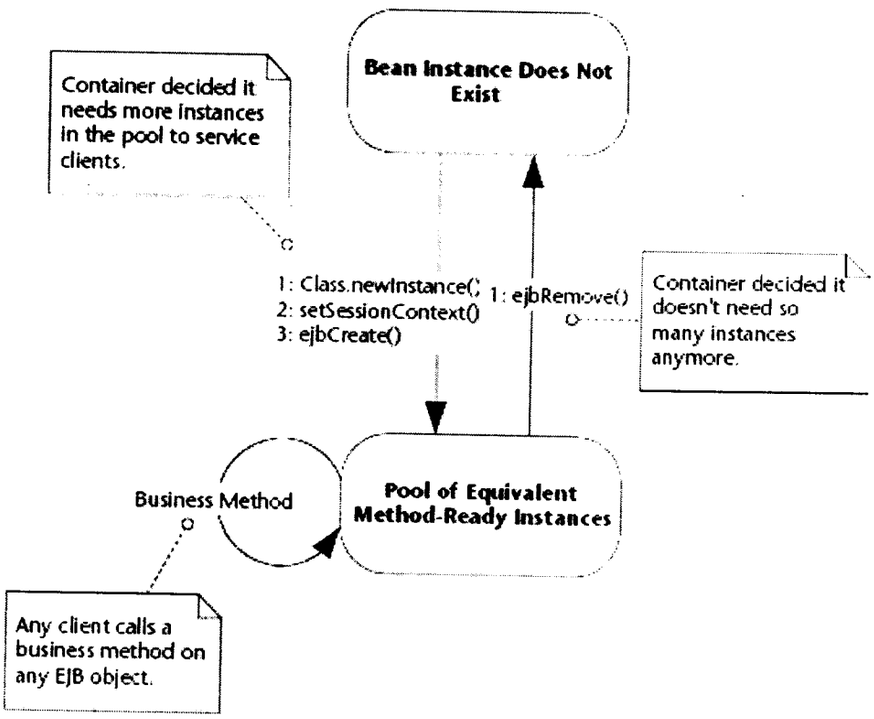


FIG. 5.4 – Cycle de vie d'un bean session stateless.
[RoSrBr05]

1. Quand le serveur d'application est lancé, aucune instance n'existe, le conteneur instantie un certain nombre de bean en se basant sur des fichiers spécifiques; ces beans sont les mêmes et peuvent être utilisés par n'importe quel client.
2. Pour créer ces instances, le conteneur appelle la méthode Class.newInstance(), suivi de l'appel de la méthode setSessionContext() pour associer un contexte au bean.
3. Après l'affectation du contexte au bean, le conteneur initialise le bean par l'appel de la méthode ejbCreate(), le client peut ainsi appeler n'importe quelle méthode qui est déléguée au bean par un EJB Object.
4. Quand le bean n'est plus utilisé, le conteneur appelle la méthode ejbRemove() pour détruire ce bean.

Quant au cycle de vie d'un bean session statefull, il est illustré par la figure 5.5 :

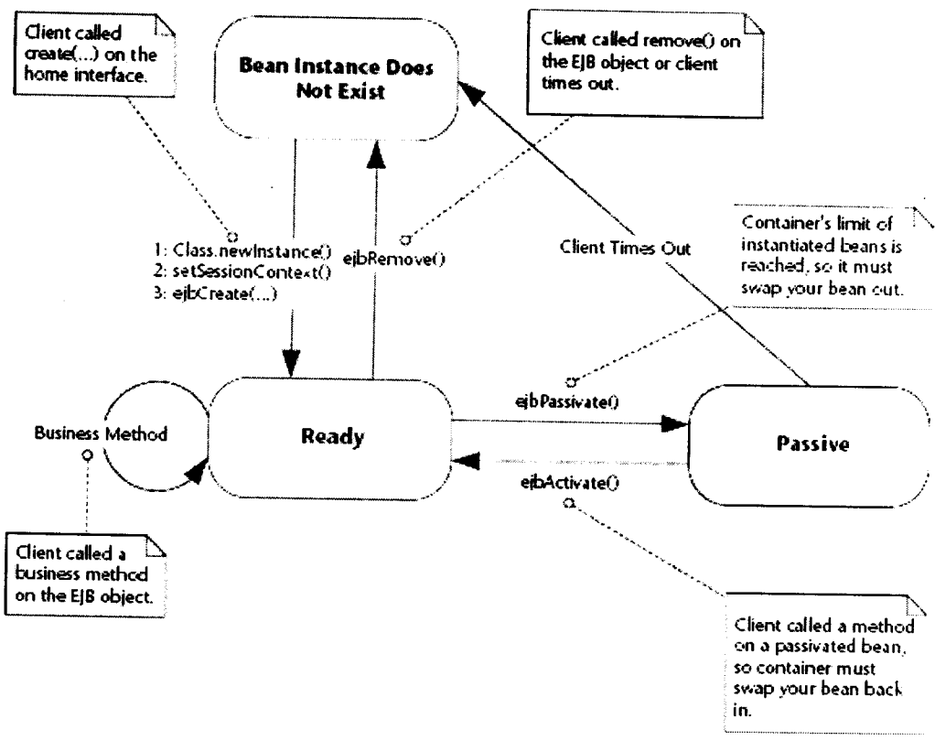


FIG. 5.5 – Cycle de vie d'un bean session statefull
[RoSrBr05]

Cette figure montre que les beans sessions statefull possèdent un cycle de vie similaire a celui des beans sessions stateless avec les différences suivantes :

1. Les instances du pool ne sont pas équivalentes du fait que chaque instance conserve un état spécifique (le conteneur ne crée un bean qu'après l'appel de la méthode create par le client).
2. utilisation des "aller-retour" pour la passivation et l'activation.

5.3 Les beans entités

On présente dans ce qui suit un autre type de composant EJB qui est un des points clés pour le développement d'applications puissantes.

5.3.1 Définition

Un bean entité est un objet persistant, c'est-à-dire il permet le stockage des données de manière permanente ; il possède un cycle de vie indépendant du client et un identifiant unique qui permet de le distinguer des autres beans.On aborde la description des beans entités par la présentation des différentes manières de persistance d'objets.

5.3.2 Concepts de la persistance

Il existe plusieurs mécanismes pour réaliser la persistance des beans entités qui sont la sérialisation des objets Java, le mapping objet/relationnel(Object Relational Mapping (ORM)) et l'utilisation des bases de données objets.

Généralement, on est confronté à la difficulté de gérer des données persistantes dans une base de données relationnelle.

Dans ce contexte, un ORM doit gérer le passage entre la représentation des données comme objet vers une représentation sous forme d'enregistrement dans une table d'une base de données.Pour cela, on peut utiliser l'API JDBC ou un produit commerciale comme oracleToopLink, Hibernate, etc....
La correspondance entre un objet et une table est montrée par la figure 5.6.

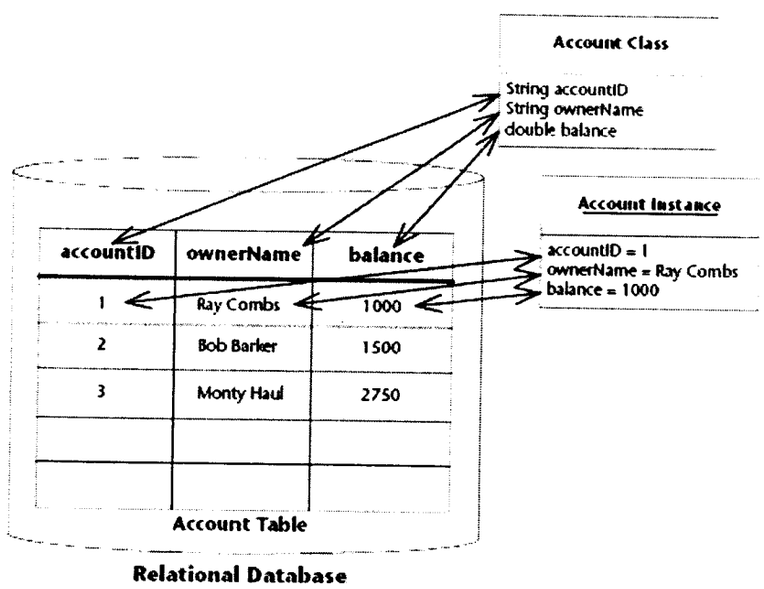


FIG. 5.6 – Exemple de mapping relationnel objet.
[RoSrBr05]

Chaque champ de la classe est lié à un champ de la table associé à cette classe.
Chaque instance de cette classe correspond à un enregistrement dans la table.
La sérialisation d'objets Java consiste à convertir un objet en format bit-blob (groupe de bits) pour qu'il puisse être stocké dans un système de persistance, contrairement aux base de données objet qui stockent directement un objet.

5.3.3 Les constituant d'un bean entité

Les beans entités contiennent l'ensemble des fichiers standards tout comme les autres types de composants EJB (les interfaces et la classe bean) avec les différences suivantes :

- *La classe bean* : elle contient l'ensemble des méthode pour gérer le bean (création, destruction, recherche,etc...) ;
- *La classe primary key* : il est recommandé d'utiliser une classe qui permet d'identifier chaque bean entité indépendamment, cette classe est très utile pour les relations complexes entre les entités.Donc une clé primaire peut être un objet contenant plusieurs attributs.

5.3.4 Caractéristiques des beans entités

Les caractérstiques des bean entités sont :

1. Ils supportent les pannes des machines et des SGBD car ils sont juste une représentation des données stockées.Si une machine tombe en panne, le bean peut être reconstruit par la création d'un nouveau bean et l'affectation des données de l'ancien bean à ce dernier.Cela veut dire que les beans entités ont un cycle de vie plus long que les beans sessions.
2. Les données d'un bean entité peuvent être manipulées directement via une base de données sans utiliser une JVM (Java Virtual Machine).Cela veut dire que le bean entité et les données sont similaires et la modification du bean implique nécessairement celle des données. Pour lire les données à partir de la base de données, le conteneur appelle la méthode ejbLoad() et pour stocker les valeurs des champs du bean, il fait appel à la méthode ejbStore().Ces méthodes ne sont pas appelées au cours d'un traitement donné.
   Ce principe est illustré par la figure 5-3 :

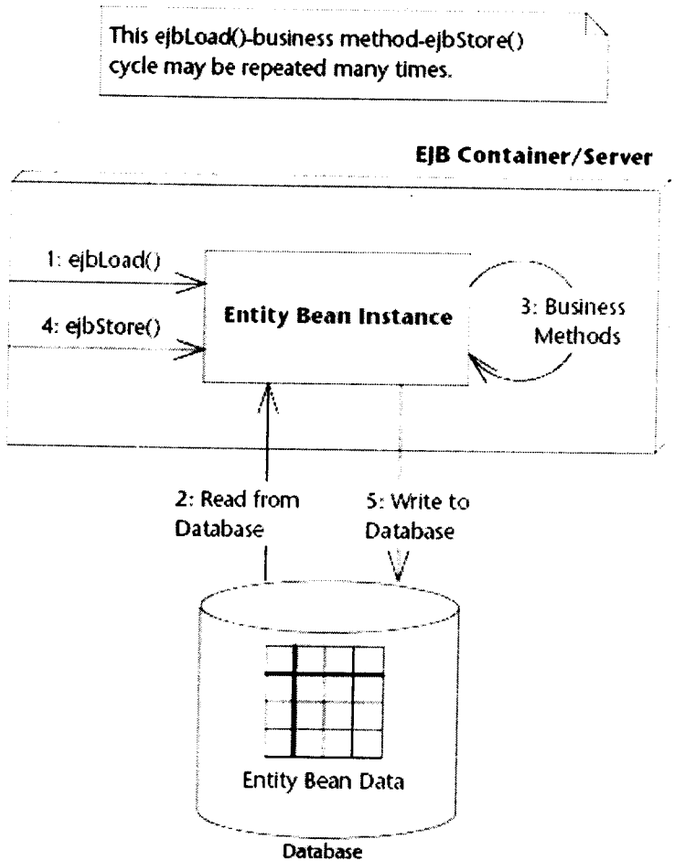


FIG. 5.7 – Chargement et stockage d'un bean entité.
[RoSrBr05]

3. On peut considérer le cas où plusieurs clients veulent accéder à une même donnée en même temps : comment le conteneur doit-il gérer cet accès pour permettre ce cas de figure ?. La réponse est que le conteneur a la possibilité de créer plusieurs instances du bean qui sont associées à une même donnée.En ce qui concerne la gestion d'accès simultané, le conteneur utilise les méthodes callbacks (ejbLoad(), ejbStore()) ainsi que les transactions (les opérations se font en tout ou rien), pour donner au client l'illusion d'accès exclusive à la donnée.Ceci est illustré par la figure 5.8 :

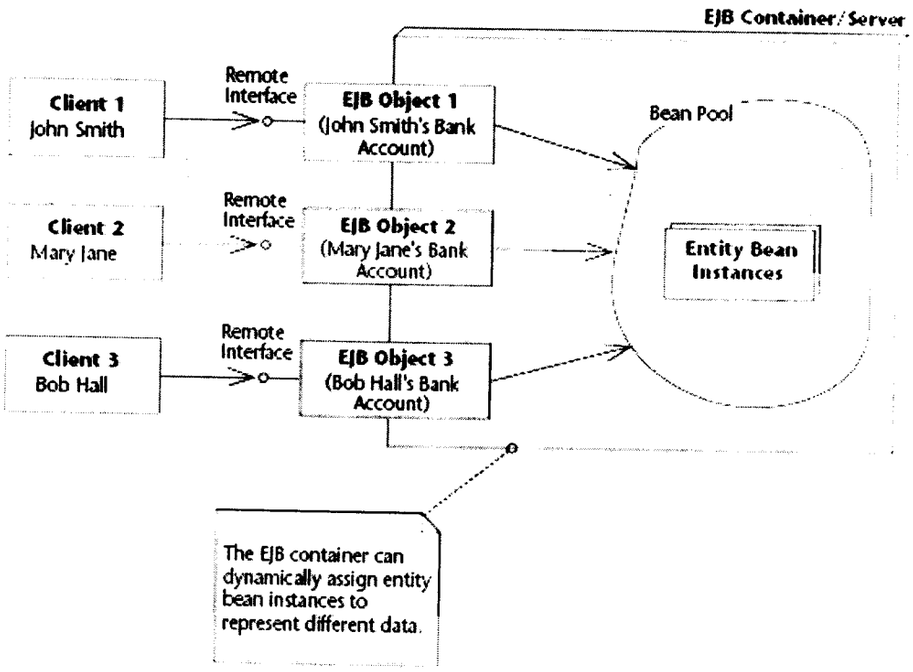


FIG. 5.8 – Le poll des beans entités.
[RoSrBr05]

4. Les beans entités sont réutilisables, c'est-à-dire qu'ils peuvent être affectés dynamiquement aux clients.Le conteneur appelle la méthode ejbPassivate() pour dissocier un bean d'un EJB Object; ce bean est mis dans le pool pour qu'il puisse être réutilisé.

5.3.5 La création et la destruction des beans entités :

Lorsque le client appelle la méthode create() par un EJBobject, le conteneur appelle la méthode ejbCreate() correspondante pour initialiser le bean et donc les données de la base.La méthode ejbCreate() retourne une primarykey qui est utilisée par le Home Object pour retourner un EJB Object.Ce processus est montré par la figure suivante :

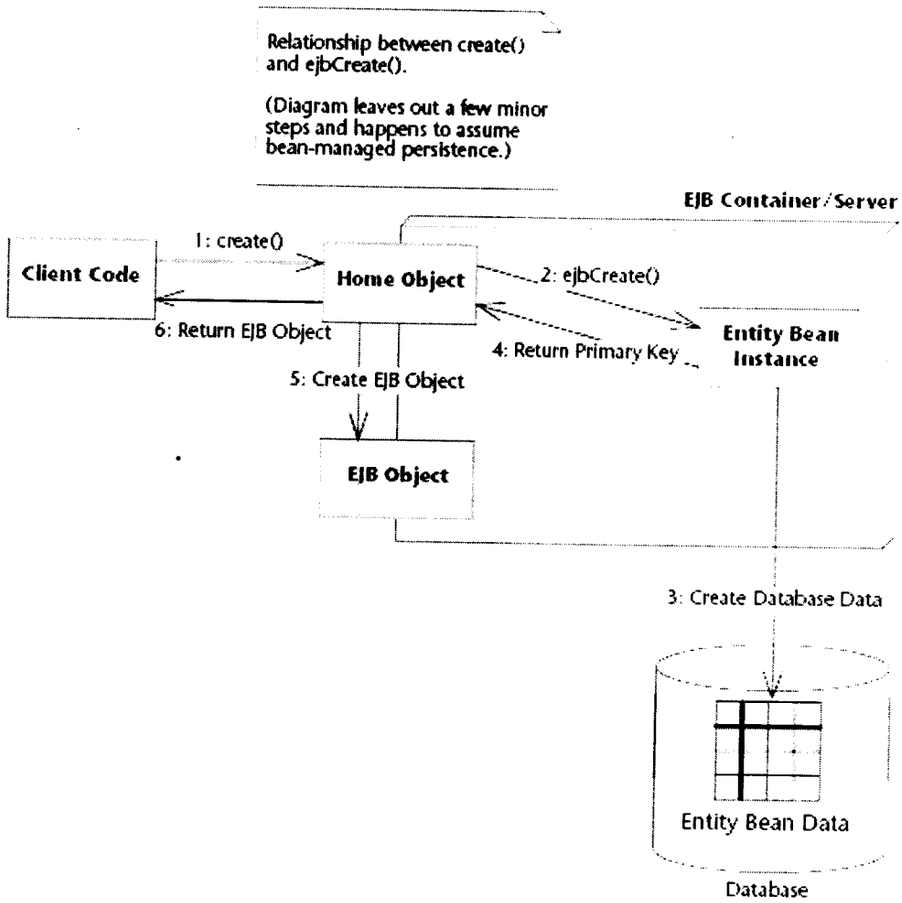


FIG. 5.9 – Création d'un EJB Object et d'un bean entité.
[RoSrBr05]

Pour supprimer les données d'une base de données, le client appelle la méthode remove() par un EJB Object ou un Home Object ; par conséquent, le conteneur fait appel à la méthode ejbRemove().Ce processus est montré par la figure suivante :

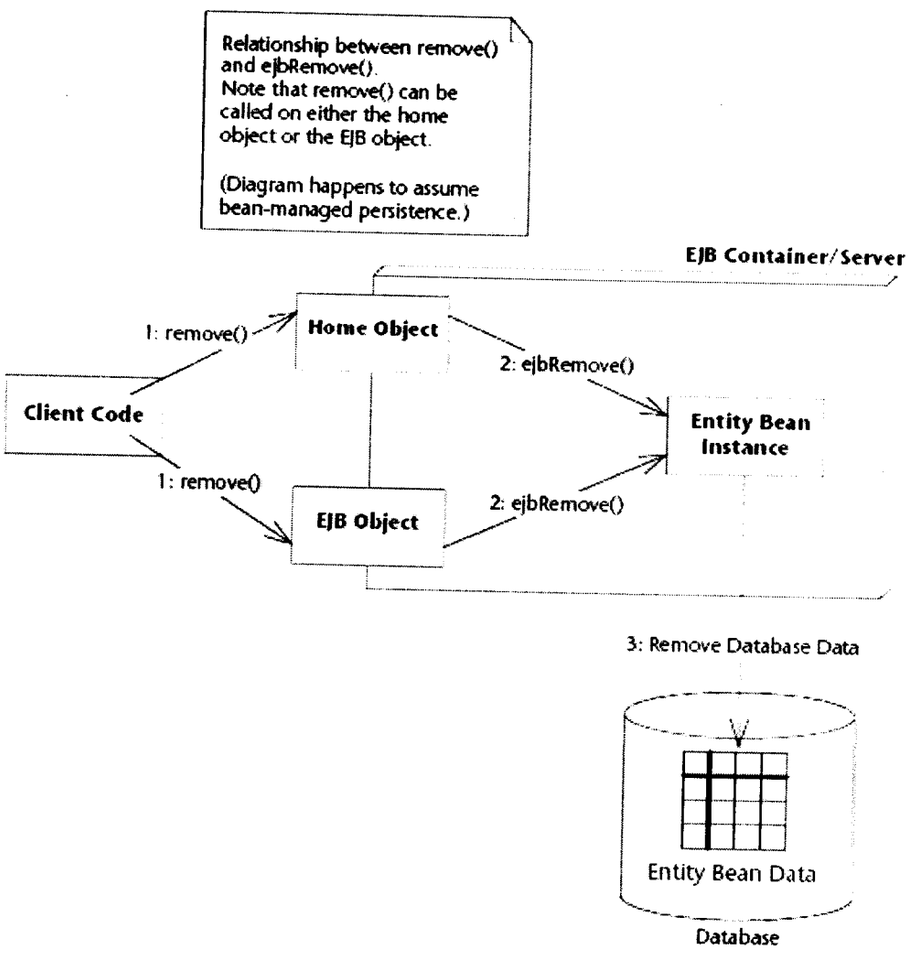


FIG. 5.10 – Suppression des données d'un bean entité.
[RoSrBr05]

Remarque : On peut effectuer des recherches sur les données en utilisant des méthodes appelées **finders** qui sont définies dans l'interface Home, ce qui différencie les beans entités des autres types de composants EJB.

5.3.6 Les différents types des beans entités

Il existe deux types de beans entités, ceux dont la persistance est assurée par le conteneur ou par le composant lui même.

1. **Les beans entités dont la persistance est gérée par le bean (Bean Managed Persistance (BMP)) :**
   - ***Définition*** : Un BMP a sa persistance implémentée par le fournisseur du bean.Ce dernier a la charge d'implémenter la logique nécessaire pour créer un nouveau bean, mettre à jour certains attributs des beans, supprimer un bean et trouver un bean dans le stockage persistant.Cela nécessite habituellement l'écriture du code JDBC pour interagir avec une base de données relationnelle ou un autre stockage persistant.Avec la gestion de persistance par le Bean, le développeur a le contrôle total de la manière dont la persistance du Bean est réalisée.
   - ***Le cycle de vie d'un BMP*** : Le cycle de vie d'un BMP est illustré par la figure suivante :

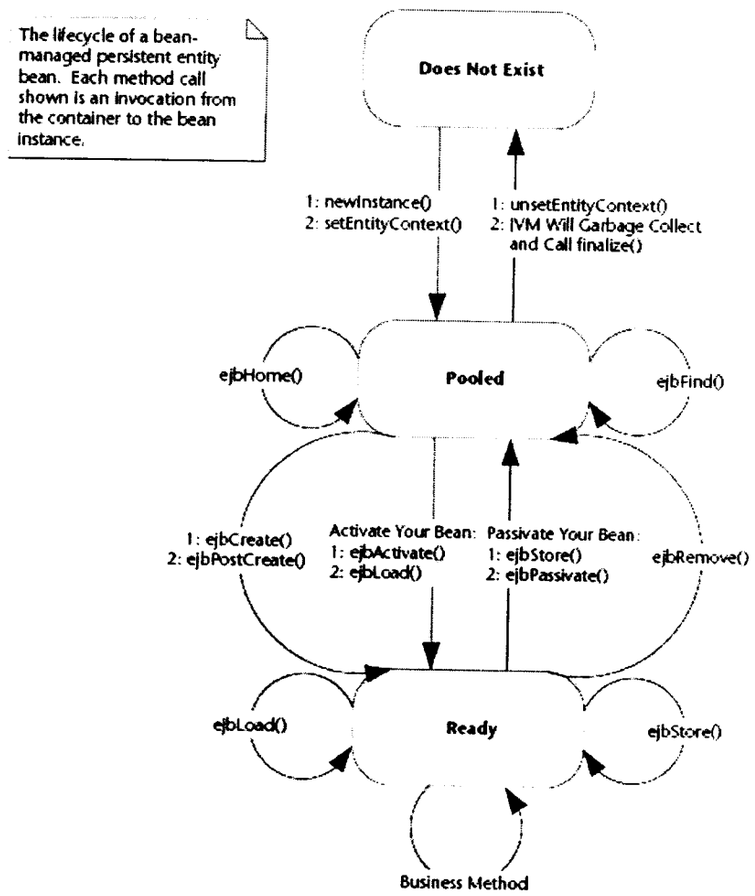


FIG. 5.11 – Cycle de vie d'un bean entité BMP.
[RoSrBr05]

- Quand le serveur d'application est lancé aucune instance n'existe ;
- Pour la création d'une nouvelle instance, le conteneur appelle la méthode newInstance(), puis il associe un contexte au bean en appelant la méthode setEntityContext() ;
- A ce niveau là, le bean est dans le pool c'est à dire il n'est associé à aucune donnée ; ce bean peut être utilisé pour effectuer une recherche dans la base de données par des méthodes finders, déjà expliquées ou pour effectuer des opérations qui ne dépendent d'aucune donnée en appelant une méthode ejbHome(), par exemple, pour connaître le nombre d'enregistrements dans la table ;
- Si le conteneur veut réduire le nombre de beans dans le pool, le bean pourra être détruit par le biais de la méthode unSetEntityContext() et le grabage Collector(ramasse-miettes) ;
- Quand un client veut créer une nouvelle donnée, il doit faire appel à la méthode create() du Home Object.Le conteneur appelle la méthode ejbCreate() d'un bean pris du pool ; il appelle ensuite la méthode ejbPosteCreate() pour compléter l'initialisation ;
- Après initialisation, le bean passe dans un état prêt, le conteneur appelle les méthodes ejbLoad() et ejbStore() périodiquement.Le bean peut appeler des méthodes métiers correspondantes à des invocations du client ;
- Lorsque le client appelle la méthode remove(), la méthode ejbRemove() est appelée par le conteneur pour supprimer les données de la base de données. le bean est retourné au pool pour qu'il puisse être réutilisé ;
- Quand un client se déconnecte, le conteneur EJB appelle la méthode ejbStore() puis la méthode ejbPassivate() pour rendre le bean dans le pool ;
- Si le conteneur veut associer un bean existant à un EJB Object (réutilisation), il appelle la méthode ejbActivate(), puis ejbLoad() pour affecter les données au bean.

2. **les beans entités dont la persistance est gérée par le conteneur(Container Managed Persistance(CMP)) :**
   - ***Définition*** : Un CMP a sa persistance assurée par le conteneur.A travers les attributs spécifiés dans le descripteur de déploiement, le conteneur fera correspondre les attributs du bean entité avec un stockage persistant (habituellement, mais pas toujours, une base de données).
     La gestion de la persistance par le conteneur, réduit le temps de développement et réduit considérablement le code nécessaire pour le composant EJB [Ecke00].
   - ***Caractéristiques des CMP*** :

- Les CMP possèdent des sous classes : la différence des CMP par rapport aux BMP est que la persistance est gérée par le conteneur, donc les CMP visent à séparer nettement la logique de persistance de la logique métier.L'avantage considérable est qu'on peut modifier le mécanisme de persistance (on peut changer une base de données relationnelle par une base de données objet) sans affecter la logique métier.Donc, pour réaliser cette séparation, la classe bean CMP n'implémente que la logique métier ; le conteneur génère une sous classe de la classe bean qui implémente la logique de persistance ;
- Les champs du bean CMP ne sont pas déclarés dans la classe bean, ils sont mis dans la sous classe par le conteneur ;
- Les méthodes get/set (méthodes permettant l'accès aux données) sont implémentées dans la sous classe ;
- La question qu'on peut se poser est comment le conteneur peut connaître les champs et les méthodes à générer ?.La réponse est que les champs et les méthodes get/set sont spécifiés dans le descripteur de déploiement. ce concept est nommé schéma de persistance abstrait ;
- Utilisation d'un langage nommé EJBQL (C'est un langage de requête orienté objet spécifique à EJB, il est similaire à SQL et utilise les instructions (SELECT, FROM, WHERE, ORDER BY)) pour la définition des méthodes finders dans le descripteur de déploiement.

3. ***Cycle de vie d'un CMP*** : Le cycle de vie d'un bean CMP est similaire à celui d'un bean BMP, l'unique différence est l'utilisation de la méthode ejbSelect() qui peut être appelée par un bean dans le pool ou dans un état prêt.Cette méthode n'est pas exposée directement au client, elle est utilisée comme méthode d'aide pour l'accès aux données.Cette méthode est caractérisé par les points suivants :
   - Elle permet d'effectuer des opérations complexes qui sont difficilement réalisable par les méthodes finders ;
   - Elle peut être utilisée pour récupérer des données depuis un autre bean. dont le cas d'une relation avec un autre bean ;
   - Tout comme les méthodes finder, cette méthode peut retourner un bean entité mais sa puissance est due au fait qu'elle peut retourner des types des champs.

   Le cycle de vie d'un bean cmp est montré par la figure 5.12 :

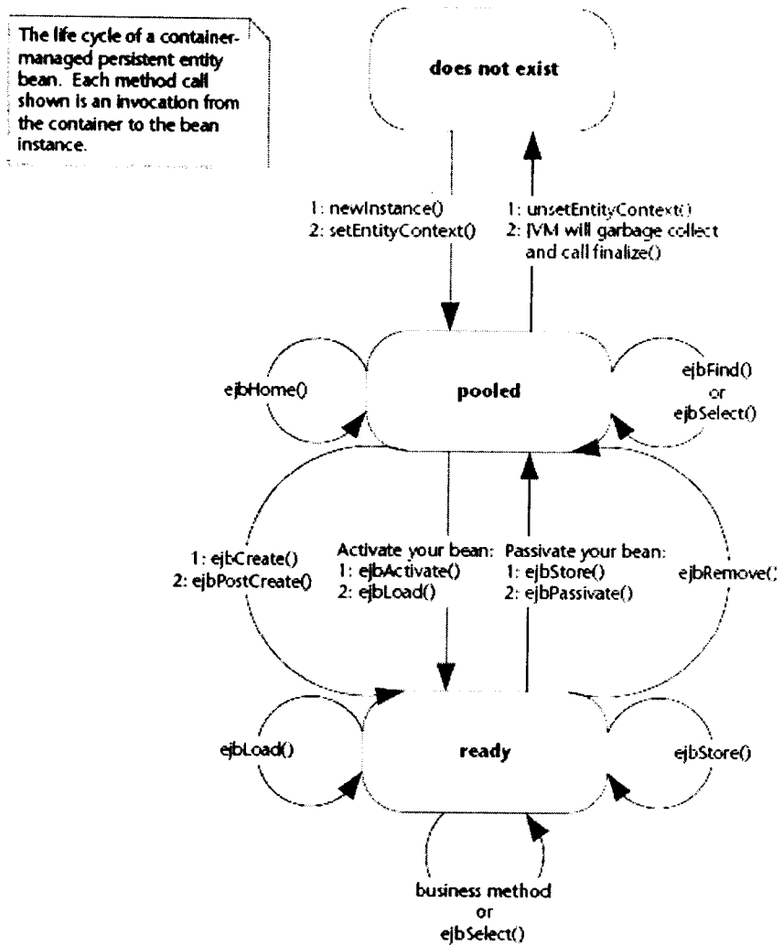


FIG. 5.12 – Cycle de vie d'un bean entité cmp.
[RoSrBr05]

5.4 Les relations entre les beans entités

Dans cette section on va présenter la notion de relation entre données, c'est à dire entre les beans. Par exemple. la relation entre étudiant et cours ou entre personne et adresse.Les relations dans les EJB sont différemment implémentée selon qu'on utilise les CMP ou les BMP ; les BMP gèrent les relations de manière explicite c'est-à-dire vous devez écrire du code pour gérer la relation ; pour les CMP, les relations sont gérées par le conteneur c'est à dire vous définissez les spécificités de la relation dans le descripteur de déploiement.

Il existe différentes caractéristiques des relations entre composants dont on cite les

plus significatives :

1. **Cardinalité :** Elle définit le nombre d'instances pouvant participer dans la relation :
   - 1 : 1 comme une relation entre un employé et une adresse.
   - 1 : N comme une relation entre un directeur et un employé.
   - M : N comme une relation entre un étudiant et un cours.

   Dans une relation 1 : 1, la clé primaire de l'une des deux tables est copié dans un champ de l'autre table ; ce champ est considéré comme clé secondaire.Si on prend un exemple d'une relation entre un employé et une société (relation 1 : N) on associe à chaque enregistrement de la table société un tableau d'employés ; cette approche est jugée non idéale.Pour cela, on définit un champ société qui contient la clé primaire de la société associée à l'employé, la figure suivante illustre ce principe :

CompanyPK	Name
12345	The Middleware Company

EmployeePK	Name	Sex	Company
20202	Ed	M	12345
20203	Floyd	M	12345

FIG. 5.13 – Relation 1 : N

   Dans le cas d'une relation M : N, une nouvelle table est créée ; cette dernière contient les deux clés primaires des deux tables.

2. **Navigabilité :** La navigabilité d'une relation spécifie la direction dont la relation navigue ; il y a deux types de navigabilités :
   - *Bidirectionnel* : on peut naviguer d'une entité A vers une entité B et de B vers A.
   - *Unidirectionnel* : on peut naviguer d'une entité A vers une entité B mais non de B vers A.

3. **Agrégation et composition :** Une relation d'agrégation est une relation d'inclusion ; par exemple, une entreprise peut contenir plusieurs véhicules, la sup-

pression d'une entreprise n'implique pas celle des véhicules.

La composition est une relation d'assemblage ou une relation d'agrégation forte : par exemple, un véhicule est composé d'un moteur, la suppression d'un véhicule entraîne celle du moteur.

5.5 Les beans orientés messages (Message Driven Bean)

5.5.1 Définition

Nouvellement introduits dans la version 2.0 de la spécification EJB, le modèle de composant message driven bean permet de développer des composants représentant des consommateurs d'événements asynchrones.Ces composants ne possèdent ni interface Home ni interface Remote.

En effet, les instances de composant message driven bean ne sont pas accédées directement par des clients de l'application.Ils sont gérés par un message listener fourni par le conteneur.

5.5.2 Caracteristiques des beans orientés messages

- Les messages drivens beans n'ont pas d'interface Home, localHome, Remote ou Local c'est-à-dire un client ne peut pas accéder à un bean à travers une interface car les messages drivens beans traitent des messages qui peuvent provenir de différents types de clients. Par exemple, un client J2EE utilise une API JMS ;
- Les messages drivens beans ne possèdent pas beaucoup de méthodes métier ;
- Les méthodes listener (méthodes permettant l'écout de message) d'un message driven bean n'ont pas de valeur de retour ;
- Les messages drivens beans ne renvoient pas d'exception aux clients ;
- Les messages drivens beans sont sans état (comme les beans sessions).

5.5.3 Cycle de vie d'un beans orientés messages

Le cycle de vie d'un message driven bean est illustré par la figure suivante :

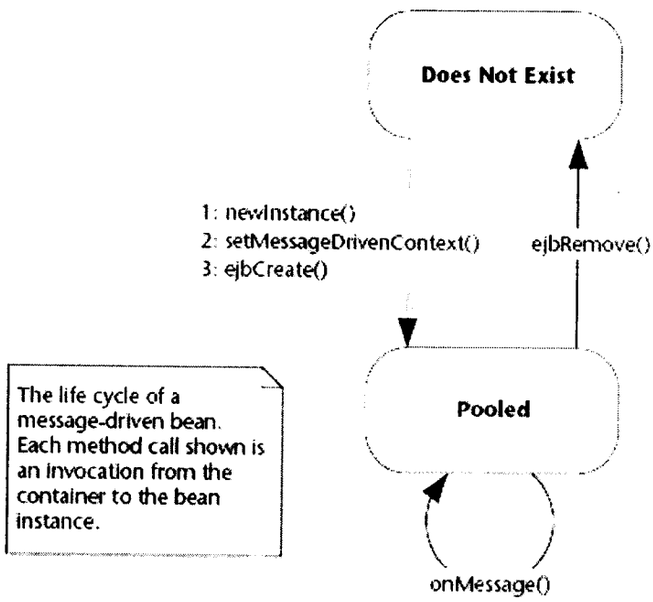


FIG. 5.14 – Cycle de vie d'un message driven bean. [RoSrBr05]

Voici la définition des différentes méthodes utilisées par les messages dirvens beans :

- *ejbCreate()* : appelée par le conteneur pour mettre le bean dans le pool ;
- *ejbRemove()* : appelée par le conteneur pour détruire le bean ;
- *onMessgae(message)* : invoquée par le conteneur pour chaque message consommé par le bean ; chaque bean consomme un seul message à la fois, mais plusieurs beans peuvent consommer plusieurs messages simultanément ;
- *setMessageDrivenContext()* : appelée par le conteneur pour affecter un contexte au bean.

5.6 Conclusion

Sun a définit trois types de composants EJB qui sont utilisés dans des contextes différents et selon les besoins.

Ce chapitre a présenté donc ces différents composants EJB en mettant en évidence leurs caractéristiques, leur cycle de vie et leur domaine d'application.

Ainsi, après avoir terminé la présentation des concepts théoriques, la prochaine partie sera consacrée à l'application et présentera les étapes importantes de l'implémentation d'une application e-commerce.

Troisième partie

Mise en ouevre d'une application e-commerce

Chapitre 6

Modélisation d'une application e-commerce

6.1 Introduction

Après avoir présenté les concepts fondamentaux de développement des composants EJB, cette partie traite d'une étude détaillée du développement et d'implémentation d'une application de commerce électronique (e-commerce), depuis la spécification jusqu'à la conception détaillée.

Avant de présenter un chapitre consacré à la partie implémentation, ce premier chapitre introduit de manière générale les applications e-commerce et leur place actuelle dans les applications web.

Aussi, nous présentons notre démarche suivie dans cette étude ainsi que les différentes technologies choisies et mises en oeuvre.Ensuite ; nous détaillons notre modélisation qui s'est basée sur UML en explicitant les différentes étapes de modélisation, chacune finalisée par un modèle.

6.2 Les applications e-commerce

6.2.1 Définition d'un site e-commerce

On appelle commerce électronique l'utilisation des réseaux informatiques ; notamment Internet pour la réalisation de transactions commerciales ; la plupart du temps il s'agit de la vente de produits sur Internet, mais le terme e-commerce englobe aussi les mécanismes d'achat par Internet.Un consommateur peut alors visiter un site e-commerce et choisir des produits.

Le e-commerce ne se limite pas à la vente en ligne mais il englobe également :

- La mise à disposition d'un catalogue électronique ;
- La réalisation de devis en ligne ;
- Le paiement en ligne ;
- Le suivi de livraison ;
- La gestion en temps réel de la disposition des produits ;
- Le service après vente.

6.2.2 Objectifs d'un site e-commerce

L'objectif principal d'un site e-commerce est de présenter les différents produits d'une entreprise dans un site web et d'offrir la possibilité de faire des commandes et de consulter ces produits.

Un site e-commerce a aussi pour objectif de permettre aux clients de consulter toutes les informations concernant les produits qui sont stockés dans une base de données.

Afin de réaliser ces objectifs, la conception de l'application doit être modulaire, extensible et bien documentée, ce qui justifie l'utilisation des diagrammes UML, des designs patterns et l'intégration des frameworks existantes.

6.3 Description de notre application

Notre application consiste à réalisé un site e-commerce d'une entreprise virtuelle doté des fonctions de base existantes dans les sites e-commerces à savoir la recherche des produits, la gestion du panier, ainsi que la gestion des commandes.

Pour cela, nous avons choisi comme type d'application la vente de matériels informatiques (PC Portables, Imprimantes, Lecteur CD/DVD, Ecrans, Disques durs), quand un internaute accède à notre site, une page d'accueil qui contient les nouveaux produits de l'entreprise s'affiche. Ainsi, un client peut effectuer des recherches rapides ou par catégorie, à l'affichage du résultat de la recherche, les produits peuvent être classés selon différents critères, le client a la possibilité de voir la fiche détaillée d'un produit qui peut être ajouté dans un panier.

Le client peut voir l'état de son panier en temps réel, il a aussi la possibilité de modifier la quantité des produits dans le panier et de voir le montant total afin de lui donner une idée sur le paiement.Le client peut aussi vider son panier.

Quand un client termine la manipulation de son panier, il peut passer à l'étape de commande ; cette étape exige l'inscription du client s'il n'est pas inscrit, sinon, il aura la possibilité de modifier son compte.

Après authentification, le client remplit un bon de commande pour qu'il passe à l'étape de paiement.

6.4 Processus de développement de l'application

Un processus définit une séquence d'étapes en partie ordonnées, qui concourent à l'obtention d'un système logiciel ou à l'évolution d'un système existant.

L'objectif d'un processus de développement est de produire des logiciels de qualité qui répondent aux besoins de leur utilisateurs dans des temps et des coûts prévisibles [RoVa04].

Lors de la réalisation de notre application, on s'est basé sur un processus de développement simple, conduit par les cas d'utilisation et centré sur l'architecture.Le processus que nous allons présenter et appliquer tout au long de ce projet est fondé sur l'utilisation d'un sous ensemble de diagrammes UML, nécessaire et suffisant.

6.5 La démarche de développement

6.5.1 Les étapes de développement d'un logiciel

Avant d'entamer l'étude de notre application, il est nécessaire de présenter les différentes étapes de développement d'un logiciel.

1. **Capture des besoins :** Cette étape consiste à spécifier les fonctionnalités du système en fonction des informations recueillies et spécifiées dans le cahier de charge, elle permet de recenser les cas d'utilisation (use cases en UML).Cette phase est importante pour le démarrage des activités d'analyse et de conception du système.
2. **Analyse des besoins :** Elle donne une description détaillée des fonctionnalisées du système à l'aide de diagrammes de séquences et met en évidence les objets et leurs liens.Au niveau de cette pahse l'accent n'est pas mis sur la façon dont le système fonctionne.
3. **Conception générique :** Cette étape permet de formaliser le comment du système c'est-à-dire de spécifier la façon dont sera conçu le système et comment il fonctionnera.
4. **Conception détaillée :** Cette étape consiste à définir précisément chaque fonctionnalité du système, elle précède la phase de codage.A ce niveau, toutes les questions relatives aux détails de la solution doivent être modélisées.Ainsi, les

interrogations restantes concernent la bonne utilisation des langages et des outils de développement.

6.5.2 Modélisation

Diagramme de cas d'utilisation :

Les cas d'utilisation décrivent les interactions de l'utilisateur avec le système(application).Le diagramme UML des cas d'utilisation est composé des acteurs et des cas d'utilisations.

Un acteur représente un rôle joué par une entité externe du système (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système [RoVa04].

1. **Identification des acteurs :** Notre application e-commerce a comme acteur principal l'internaute.
2. **Identification des cas d'utilisation :** les cas d'utilisation sont identifiés à partir des acteurs du sytème.Ainsi ; les cas d'utilisation principaux sont :
   - Rechercher des produits ;
   - Gérer son panier ;
   - Effectuer une commande ;
   - Consulter une commande.

D'où le modèle UML ci-après :

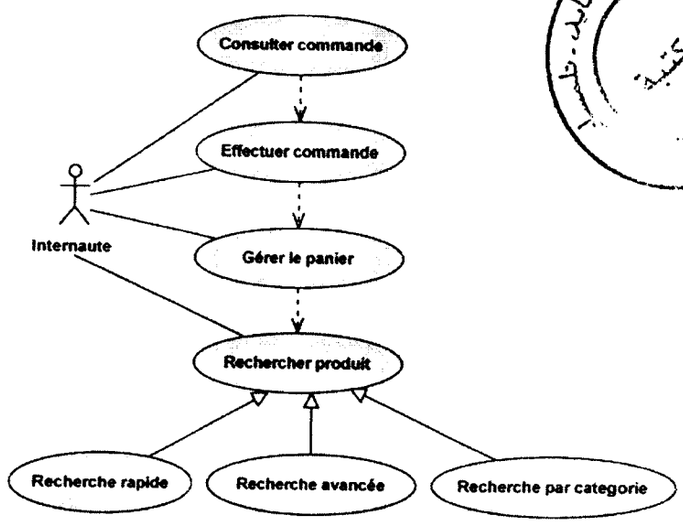


FIG. 6.1 – Diagramme de cas d'utilisation.

Ce modèle est accompagné d'une description textuelle de chaque cas d'utilisation afin d'obtenir une expression des besoins précise et décrire les differents scénarios.

1. **Rechercher des produits :**
   - ***acteur principal*** :Internaute
   - ***Objectifs*** :L'internaute veut trouver un produit le plus rapidement possible.
   - ***Préconditions*** :le catalogue est disponible
   - ***Scénario*** :
     (a) L'internaute lance la recherche à partir d'un mot clé.
     (b) Le système lui affiche les résultats qui peuvent être triés par prix.On y trouvera d'autres classifications.
     (c) Sélectionner le produit.
     (d) Le système affiche la fiche détaillée.
     (e) L'internaute peut éventuellement mettre le produit dans son panier.
   - ***Extensions*** :
     (a) L'internaute choisit d'effectuer une recherche par catégorie.
     (b) Le système n'a pas trouvé de produit, il signale l'echec et il propose à l'internaute d'effectuer une nouvelle reherche.
     (c) Au cas où le système trouve beaucoup de produits, le système indique le nombre de produits trouvés ; il affiche les produits sur différentes pages.
2. **Gérer son panier :**
   - ***acteur principal*** :Internaute
   - ***Objectifs*** :L'internaute doit avoir la possibilité d'enregistrer un produit dans un panier virtuel.
   - ***Scénario :***
     (a) L'internaute enregistre le produit dans le panier.
     (b) L'internaute demande accès à son panier.
     (c) Le système lui affiche l'état de son panier.
     (d) L'internaute valide son panier en demandant à effectuer une commande
   - ***Extensions*** :
     (a) Le panier est vide, le système affiche un message d'erreur à l'internaute et lui propose de revenir à une recherche de produit.
     (b) L'internaute modifie les quantités des lignes du panier ou en supprime et revalide le panier en demandant le recalcul du total.
     (c) L'internaute peut effectuer une nouvelle recherche.

3. **Effectuer commande** :
   - ***acteur principal*** : Internaute
   - ***Objectifs*** : Permettre à l'internaute l'accès au formulaire de bon de commande
   - ***Préconditions*** :
     - Le panier n'est pas vide.
     - L'internaute doit être abonné.
   - ***Scénario*** :
     (a) La saisie des informations nécessaires au paiement et à la livraison.
     (b) Le système affihce un récapitulatif.
     (c) Le système valide la commande.
     (d) Le système envoie la commande au service client.
     (e) le système confirme la prise de commande de l'internaute.
   - ***Extensions*** :
     (a) L'internaute est déjà client(il doit s'authentifier par e-mail et mot de passe).
     (b) S'il n'est pas client ,il doit créer un compte pour pouvoir accéder au bon de commande.

Diagramme de séquence

Nous utilisons le terme diagramme de séquence système pour souligner le fait que le système est considéré comme une boite noire.

Le comportement du système est décrit, vu de l'extérieur, sans préjuger de sa forme de réalisation.

Les cas d'utilisation décrivent les interactions des acteurs avec le site web que nous voulons concevoir.

Ces interactions se font par envoi de messages ; ces messages sont représentés sous forme de diagramme de séquence UML.

Donc, un diagramme de séquence est la description détaillée d'un cas d'utilisation. Voici les diagrammes de séquences corespondent à chaque cas d'utilisation :

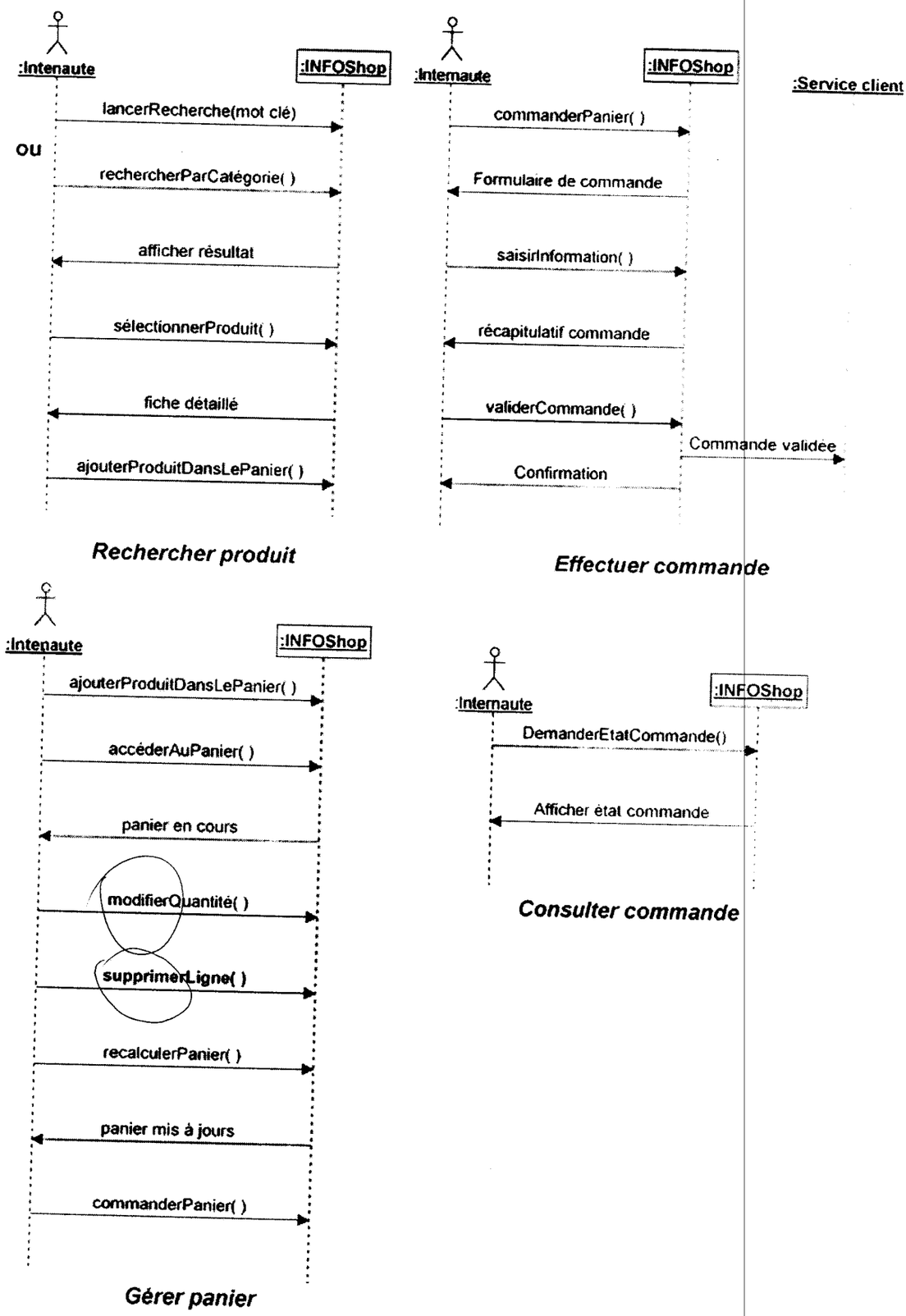


FIG. 6.2 – Diagrammes de séquence.

6.5.3 Diagramme de classes d'analyse

Après avoir élaboré le diagramme des cas d'utilisation, nous allons établir un diagramme initial de classes d'analyse UML.Ce dernier contient l'ensemble des classes constituant l'application. Avant de donner le modèle, il est nécessaire d'identifier les classes du système.Pour ce faire,nous allons procéder par cas d'utilisation.

1. **Rechercher des produits** : Il n'y a qu'une seule classe concernée qui est "produit".
2. **Gérer son panier** : Nous avons comme classe "panier", "ligne panier" et "produit".
3. **Effectuer commande** : Nous avons comme classe "commande", "client", "panier" ,"carte bancaire" et "adresse".

Un panier peut être composé d'une ou plusieurs ligne panier.
Les classes pc portable, imprimante, ecran ,dique dur, lecteur cd et lecteur dvd héritent de la classe produit.

Après avoir combiné les classes entre elles et les relations ,nous obtenons le modèle suivant :

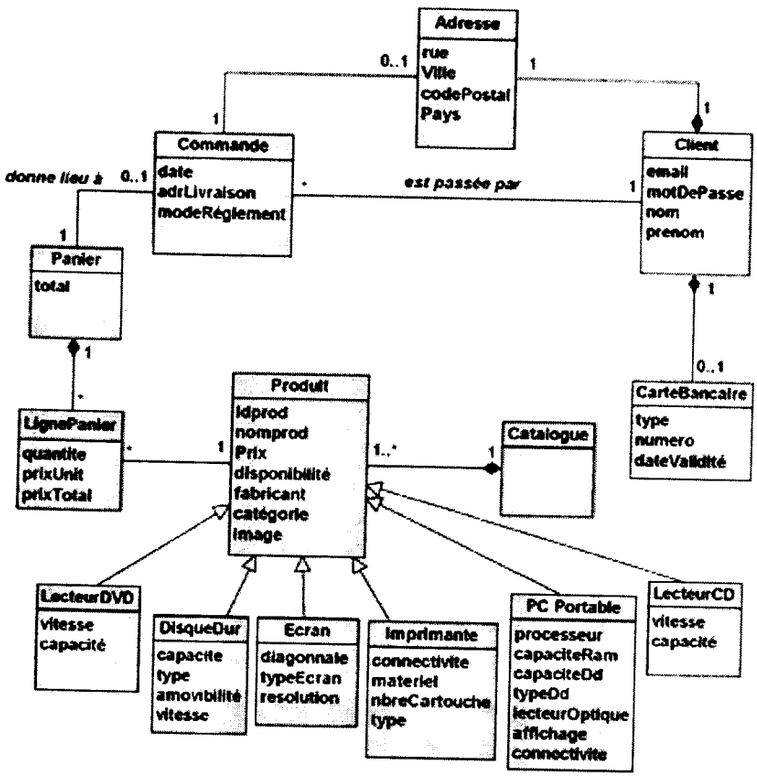


FIG. 6.3 – Diagramme de classes d'analyse.

Diagramme de classes participantes

pour compléter l'étude de la phase d'analyse, nous allons utiliser un diagramme de classes spécifique appelé diagramme de classes participantes qui consiste à identifier l'ensemble des classes réalisant les traitements vis-a-vis des cas d'utilisation (les principales classes d'IHM (Interface Homme Machine) et les classes qui décrivent l'application).

Démarche : Il existe trois types de classes à identifi1er :

1. Les classes qui permettent les interactions entre le site web et les utilisateurs.Elles sont qualifiées de "Dialogue".
2. Le deuxième type contient la logique de l'application et il est qualifié de "Controle".Il fait la transition entre le dialogue et les entités.
3. Le troisième type représente l'ensemble de classes qui décrivent les entités du système.Il est qualifié de "Entité".Ce dernier provient du modèle de classes d'analyse déjà élaboré.

La fusion de ces trois types de classes avec leurs relations, donnera ce qu'on appelle diagramme de classes participantes. Notre démarche pour élaborer ce modèle est de procéder par cas d'utilisation.

1. **Rechercher des produits :**

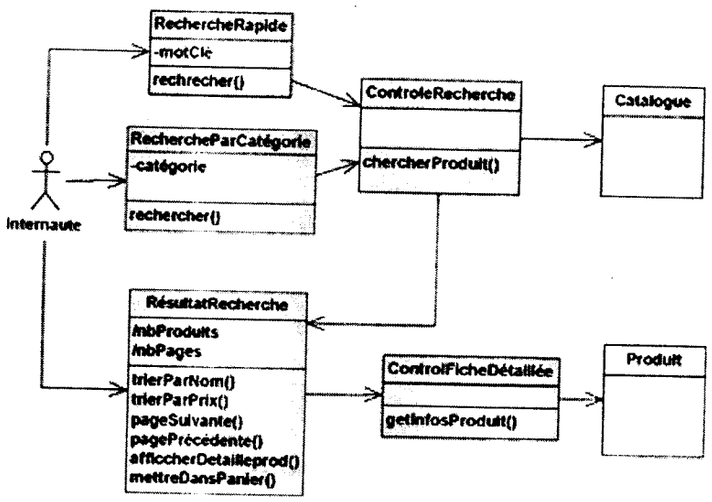


FIG. 6.4 – Diagramme de classes participantes (rechercher des produits).

2. **Gérer son panier :**

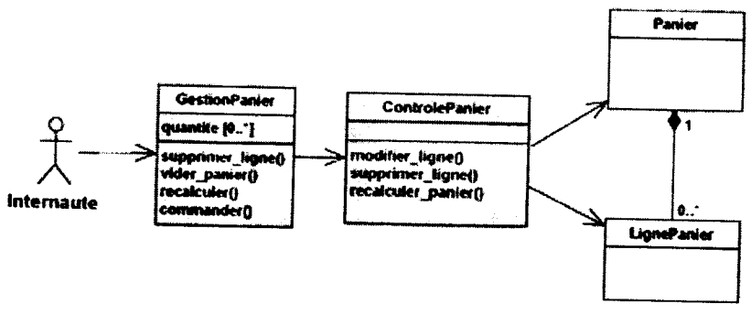


FIG. 6.5 – Diagramme de classes participantes (gérer son panier).

3. **Effectuer commande :**

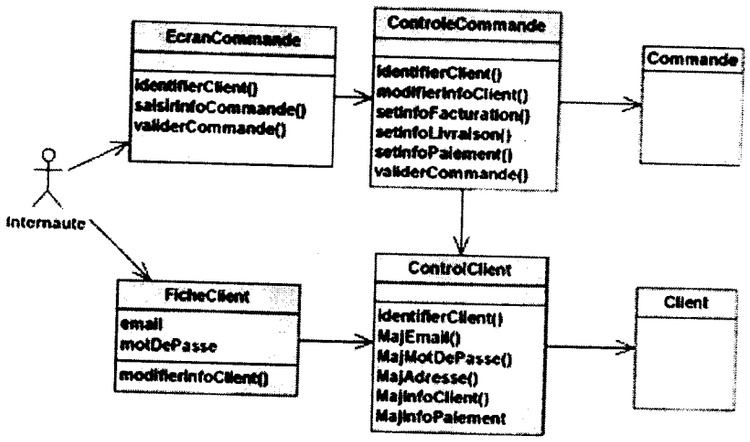


FIG. 6.6 – Diagramme de classes participantes (effectuer commande).

Diagramme d'activités

Un diagramme d'activités illustre la navigation dans le site web, nous allons nous servir d'un certain nombre d'éléments standards pour réaliser ce diagramme ; en effet, un diagramme d'activités contient un nombre restreint d'éléments qui sont :

- des activités ;
- des transitions entre activités, pouvant porter des conditions ;
- des branchements conditionnels ;
- un début et une ou plusieurs terminaisons possibles.

Voici le diagramme d'activités :

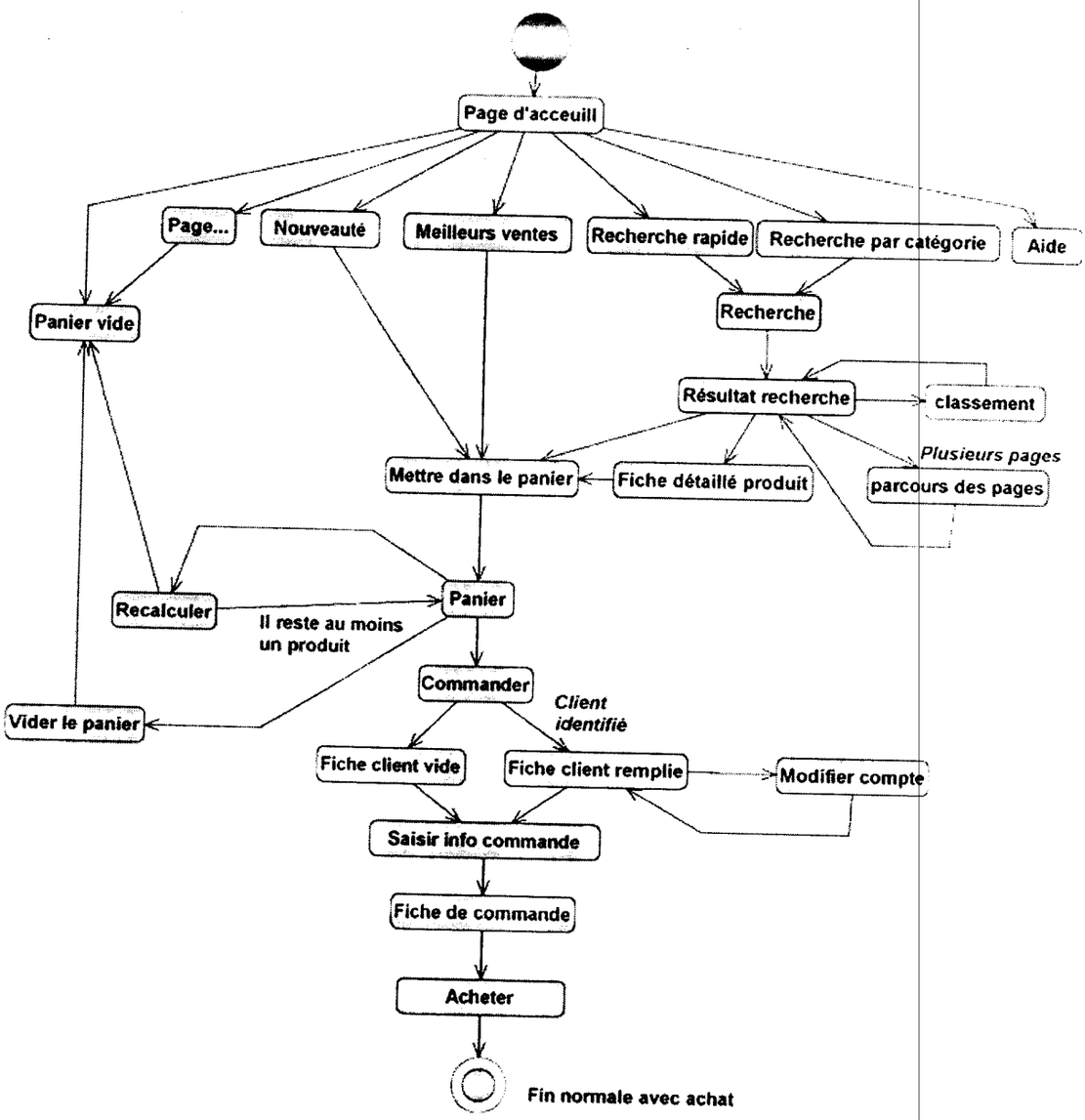


FIG. 6.7 – Diagramme d'activités.

Diagramme de classes de conception préliminaire

Après avoir identifié les cas d'utilisation, et poursuivi leur description détaillée grâce aux diagrammes de séquence complétés par des diagrammes des classes participantes, on passe maintenant à la conception, nous allons répartir tout le comportement du système entre les classes d'analyse.Voici les diagrammes des classes de conception préliminaire.

1. **Rechercher des produits :**

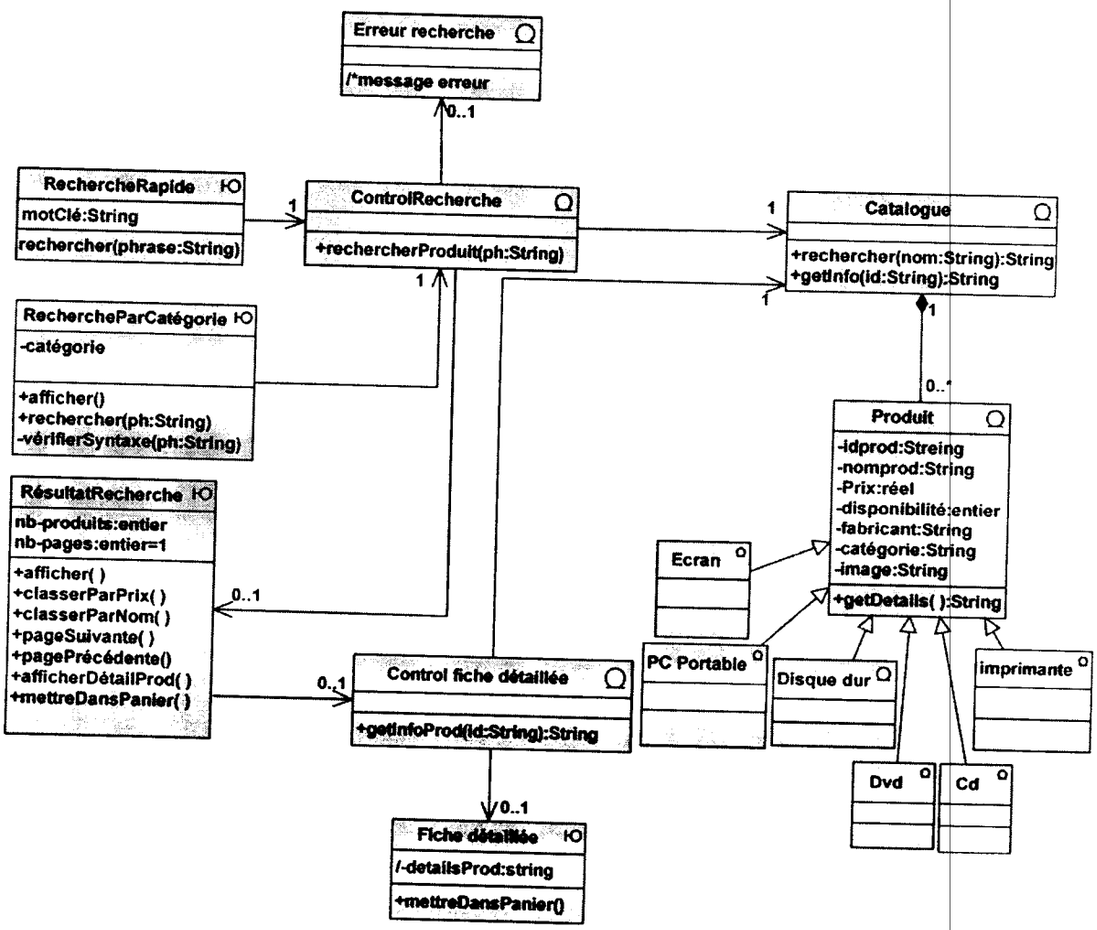


FIG. 6.8 – Diagramme de classes de conception préliminaire (rechercher des produits).

2. **Gérer son panier :**

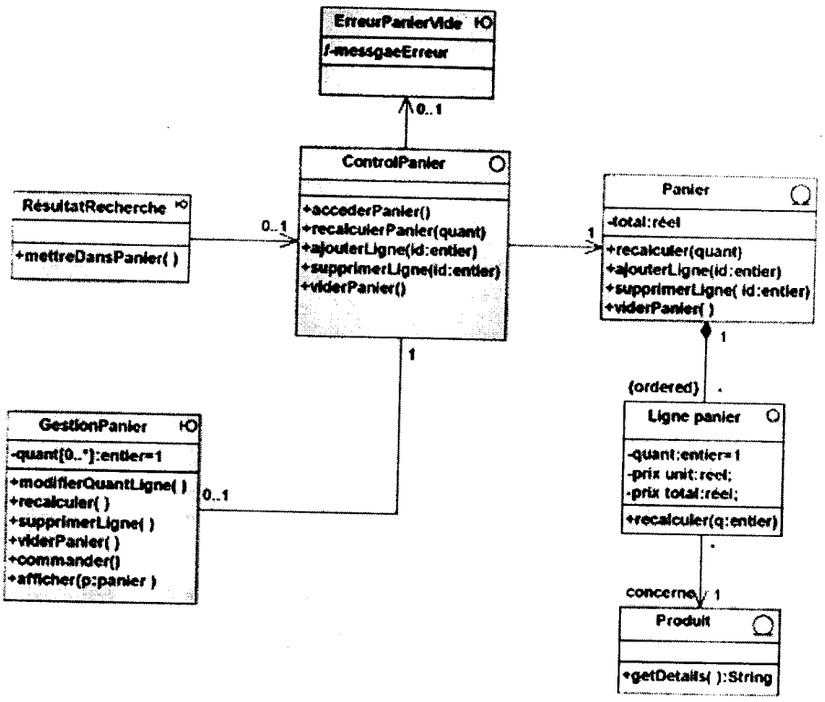


FIG. 6.9 – Diagramme de classes de conception préliminaire (gérer son panier).

3. **Effectuer commande :**

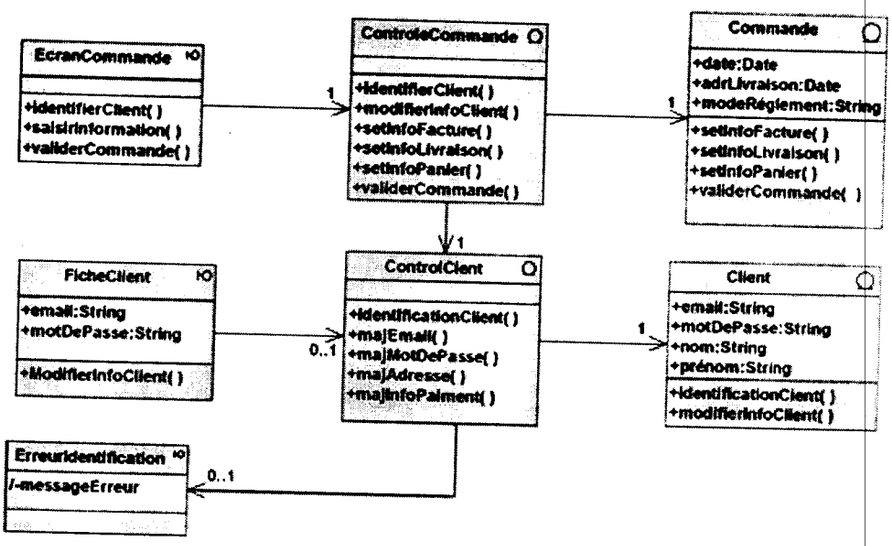


FIG. 6.10 – Diagramme de classes de conception préliminaire (effectuer commande).

Diagramme de classes de conception détaillée

Restituons cette dernière activité de conception dans l'ensemble du processus décrit dans les cas d'utilisation.Il s'agit d'affiner ce que nous avons réalisé, de façon générique, dans le diagramme précédent.C'est-à-dire que nous allons maintenant incorporer dans les diagrammes l'architecture et les choix technologiques qui vont modifier les classes de conception préliminaire, les préciser, ajouter de nouvelles classes plus techniques, etc....

N'oublions pas l'objectif principal du modèle de conception détaillée : il peut être traduit directement en code ! [Roqu03] Avant de présenter les différent modèle, voici l'architecture technique de notre application décrite par la figure 6.11 :

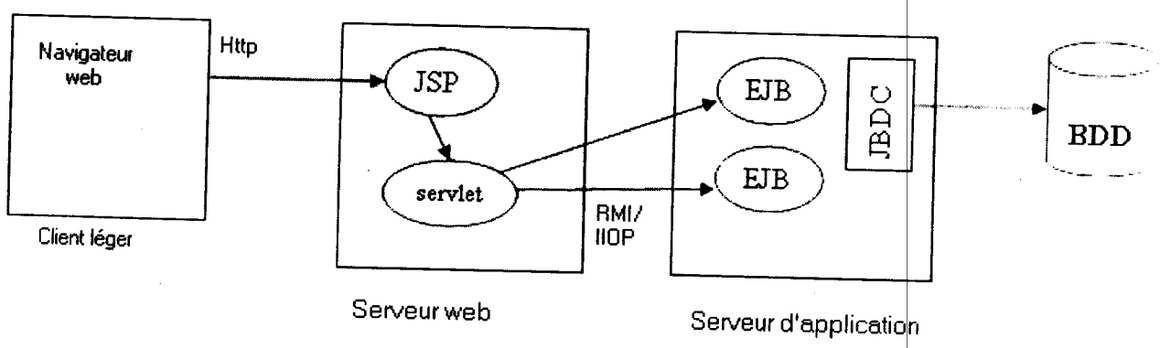


FIG. 6.11 – L'architecture technique de l'application.

- **Le client :** c'est un simple navigateur web.Il peut soumettre des requêtes HTTP vers le serveur web.
- **Le serveur web :** déclenche les traitements côté serveur, et génère les résultats sous forme de page HTML ; les pages web hébergées par le serveur web peuvent être statique (pages HTML) ou dynamique (pages JSP).
- **Le serveur d'application :** c'est l'exécuteur principal de la logique métier coté serveur.Il se concentre sur la mise en oeuvre des composants EJB ; il peut être sollicité par le serveur web à travers le protocole RMI/IIOP.
- **Serveur de données (SGBD) :** permet de gérer la persistance des données.L'accès à ces données peut être réalisé grâce aux API offertes comme JDBC, ou en ajoutant une couche intégration pour réaliser le mapping objet relationnel. Pour notre application, nous avons utilisé une base de données relationnelle et l'API JDBC.

Voici les diagrammes de classes de conception détaillée :

1. **Rechercher des produits :**

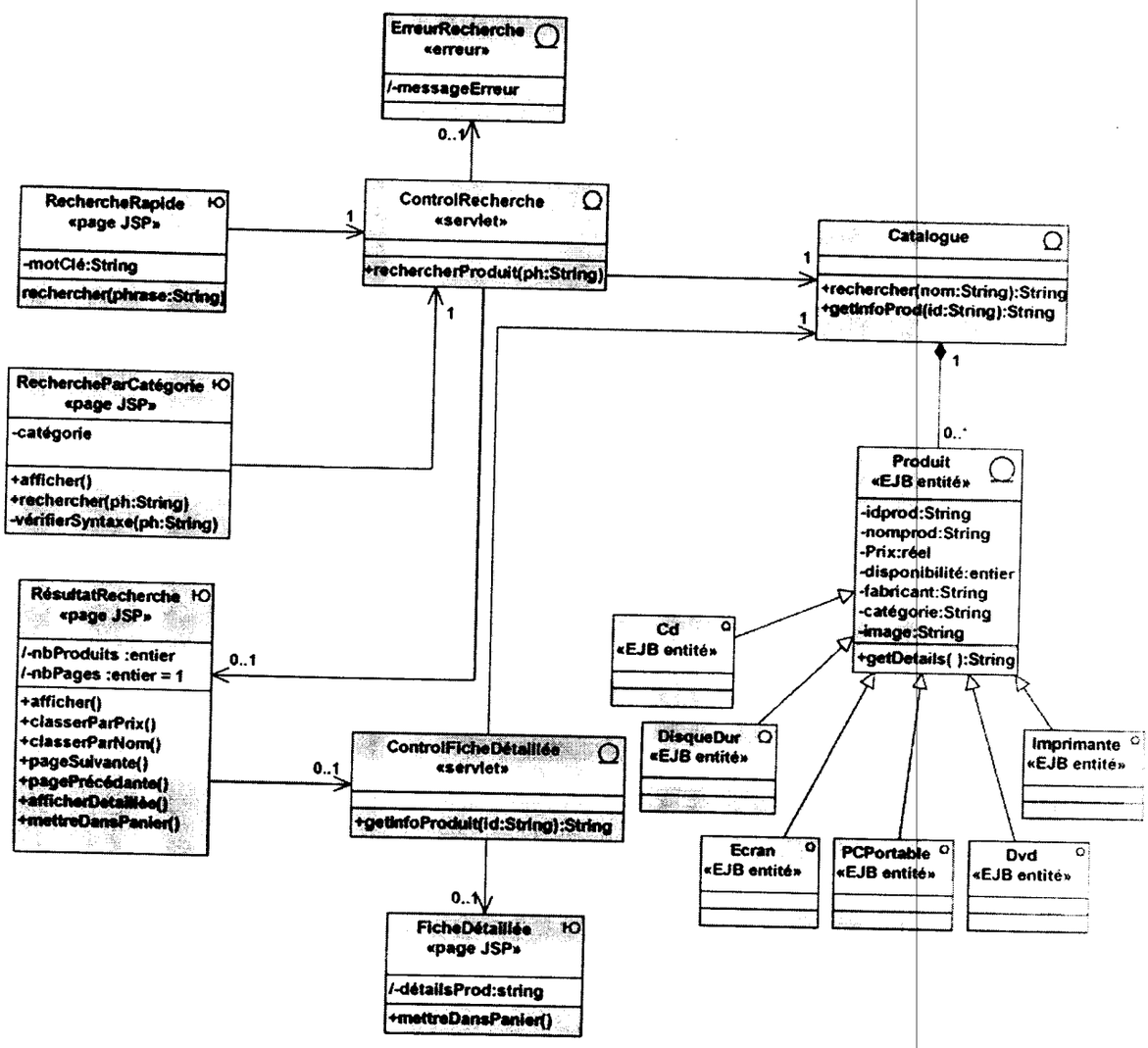


FIG. 6.12 – Diagramme de classes de conception détaillée (rechercher des produits).

2. **Gérer son panier :**

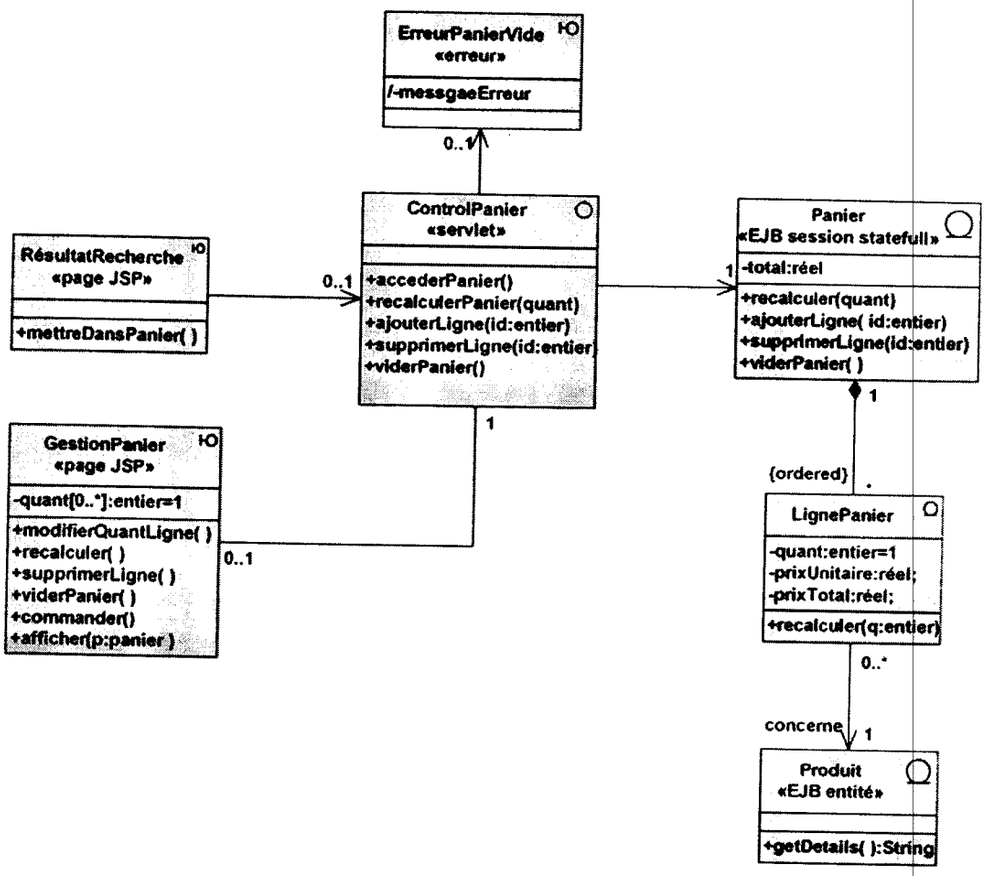


FIG. 6.13 – Diagramme de classes de conception détaillée (gérer son panier).

3. **Effectuer commande :**

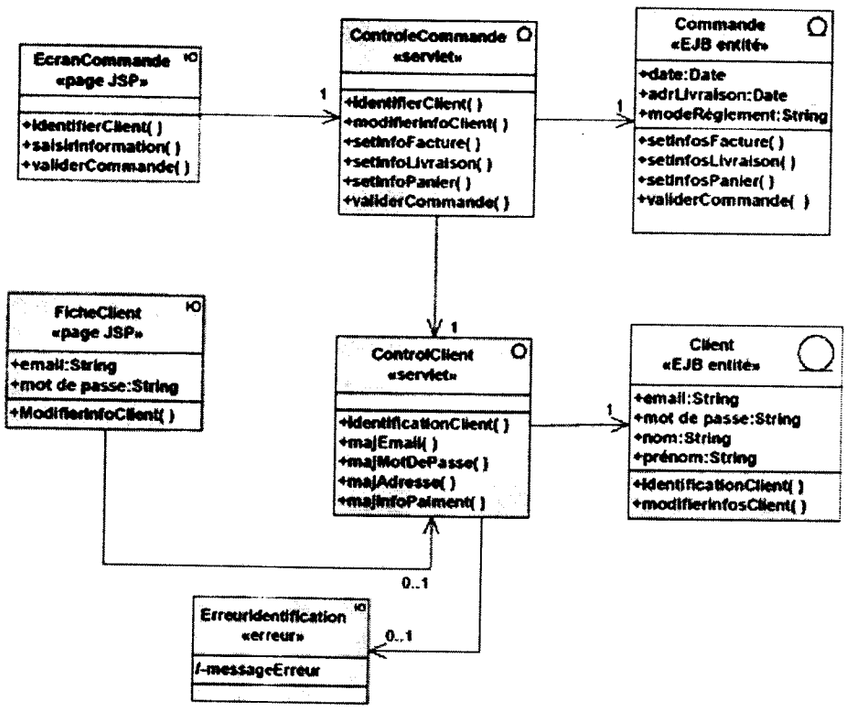


FIG. 6.14 – Diagramme de classes de conception détaillée (effectuer commande).

6.6 conclusion

L'objectif principal de ce chapitre est de présenter l'étape de modélisation de notre application.On a essayé de faire la liaison entre les différents diagrammes UML obtenues, on a aussi présenté la structure générale de notre application.

Ce chapitre se présente comme une introduction au chapitre suivant dans lequel on va donner les grandes fonctionnalités de notre application, en présentant des exemples de code d'implémentation.

Chapitre 7

Implémentation d'une application e-commerce

7.1 Introduction

Après avoir présenté l'architecture générale de notre application ainsi que les différents diagrammes UML ; la dernière phase de développement constitue l'implémentation effective de l'application dans un langage de programmation, tout en respectant bien évidemment les modèles de conception.

Ce chapitre présente une description détaillée de notre application ainsi que les différents outils utilisés pour sa réalisation.

7.2 Implémentation

Dans cette section, nous présentons les fonctionnalités de notre application, mais avant tout, il est important de présenter les différents outils utilisés pour le développement de notre application.

7.2.1 Outils de développement :

1. **Environnement de Développement Intégré (EDI)** :Pour l'implémentation de notre application, on a choisi de travailler avec l'environnement JBuilder de Borland, version 2005, car il facilite le développement des EJB grâce à un outil graphique pour concevoir les composants EJB.Ces derniers sont organisés sous forme de modules EJB.JBuilder contient aussi un module Web pour développer de manière aisée les pages JSP et les Servlets.Il offre aussi la possibilité de travailler avec plusieurs serveurs d'applications comme JBoss, WebSphere, We-

bLogic ....Il permet aussi l'intégration de serveurs Web comme Tomcat. Ainsi JBuilder se présente comme un outil très puissant pour le développement des applications distribuées.

2. **Le serveur d'applications :** Pour le choix d'un serveur d'application nous avons opté pour le serveur JBoss.Ce choix est justifié par le fait qu'il est open source, et qu'il dispose d'un conteneur EJB très puissant.Il est considéré comme l'un des meilleurs serveurs d'applications du marché.
3. **Le serveur web :** Nous avons utilisés le serveur web Tomcat issue de la communauté open source Apache, pour fournir un environnement d'exécution de JSP et Servlets.
4. **Le serveur de données :** Le SGBD que nous avons utilisé est MYSQL.

7.2.2 Description des fonctionnalités de notre application

Comme déjà présenté au chapitre précédent, notre application web a pour objectif de fournir un système d'achat de produits sur internet.Notre site e-commerce est nommé InfoShop et il met en oeuvre les fonctions de base à savoir la recherche des produits, la gestion du panier et la gestion des commandes. Au lancement de note site, la page suivante s'affiche :

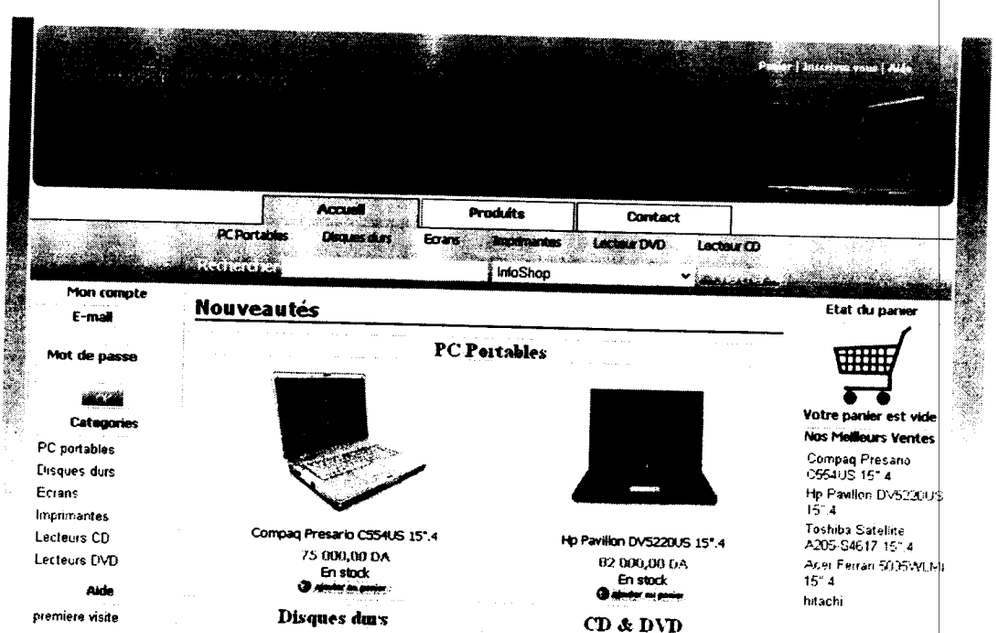


FIG. 7.1 – Page d'accueil.

L'internaute peut effectuer des recherches par mot clé ainsi que des recherches par catégorie.

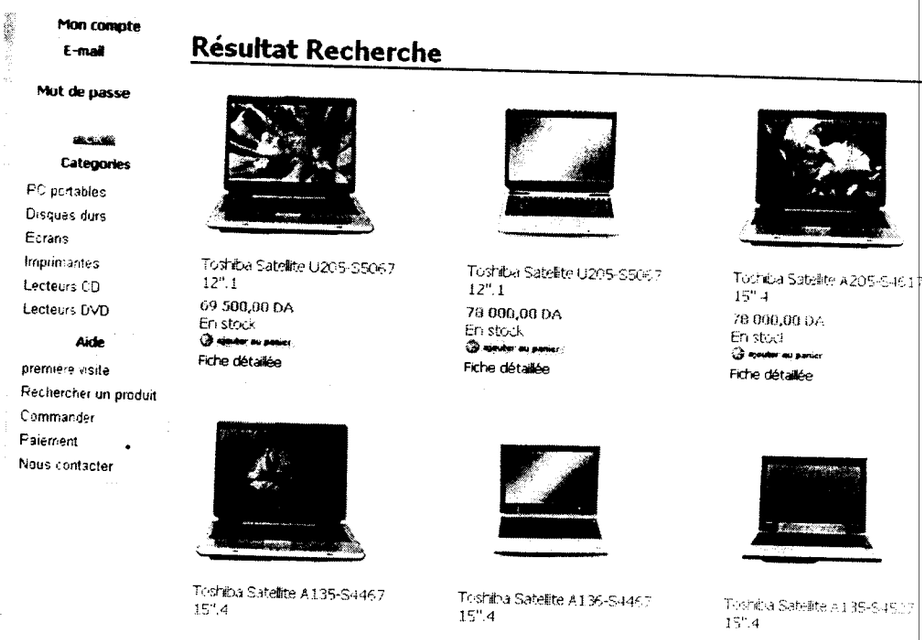


FIG. 7.2 – Résultat de recherche.

Le résultat est affiché dans une ou plusieurs pages selon le nombre de produits trouvés.L'internaute peut les visualiser en consultant les différents pages.

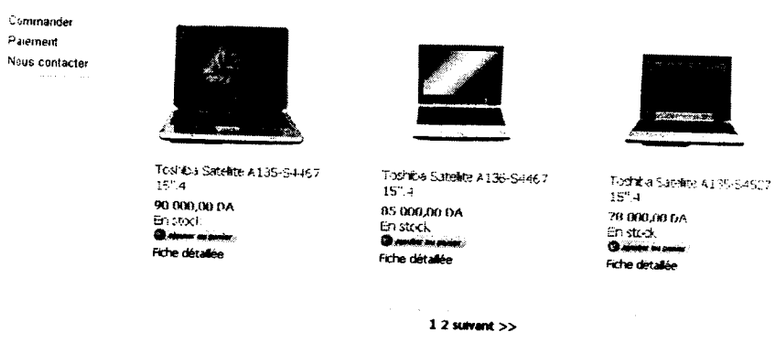


FIG. 7.3 – Résultat de recherche.

L'internaute peut voir la fiche détaillée d'un produit.

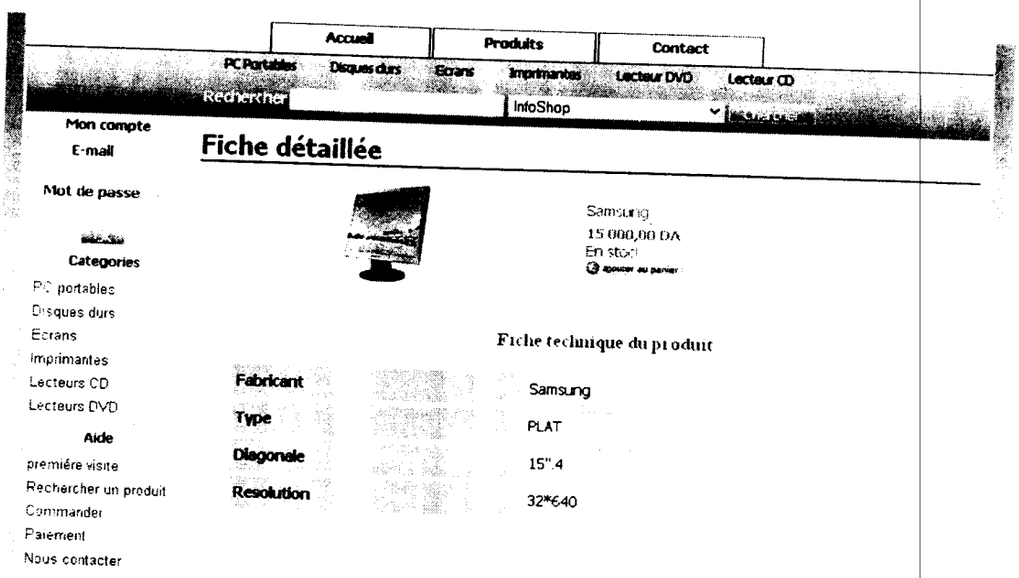


FIG. 7.4 – Fiche détaillée.

L'internaute peut ajouter un produit dans son panier.

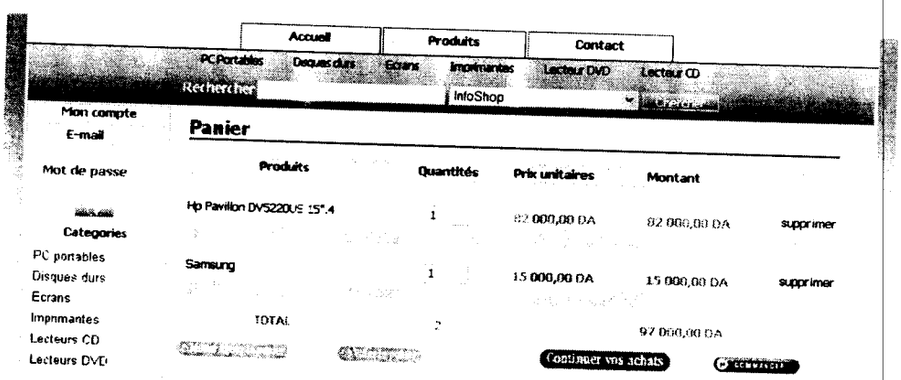


FIG. 7.5 – Consulter panier.

L'internaute peut continuer ses achats, comme il peut effectuer différentes opérations sur son panier.Il a la possibilité de supprimer une ligne panier figure7.6, de modifier la quantité des produits à acheter et de mettre à jour son panier figure7.7.

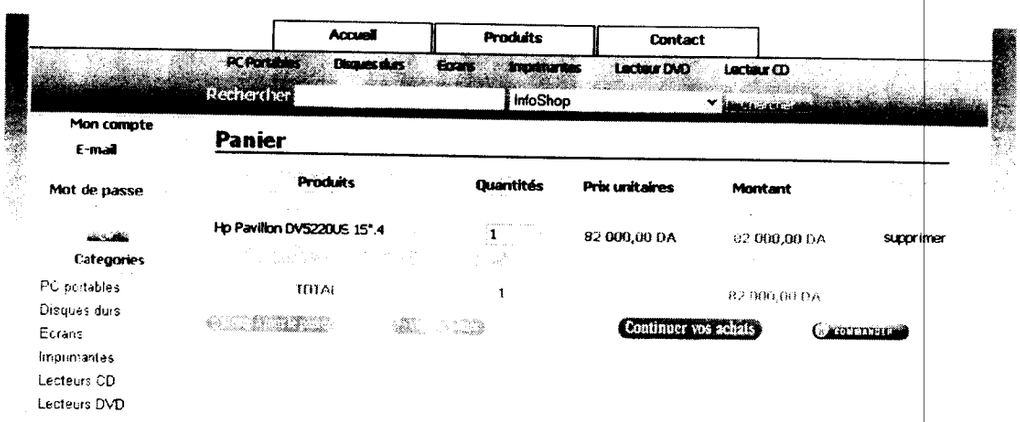


FIG. 7.6 – Supprimer un produit du panier.

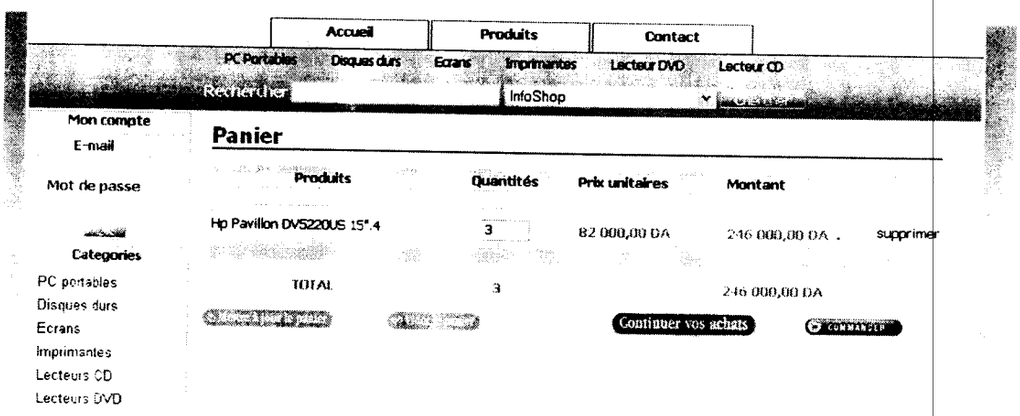


FIG. 7.7 – Modifier quantité.

Quand l'internaute valide ces achats, il passe à l'étape de commande.Dans cette étape deux cas se présentent :

1. **L'internaute est déjà client :** Dans ce cas il saisie son e-mail et son mot de passe, le client aura la possibilité de modifier ses informations pour la livraison.
2. **L'internaute n'est pas client :**Dans ce cas il doit remplir un formulaire contenat toute les informations nécessaires à son authentification.

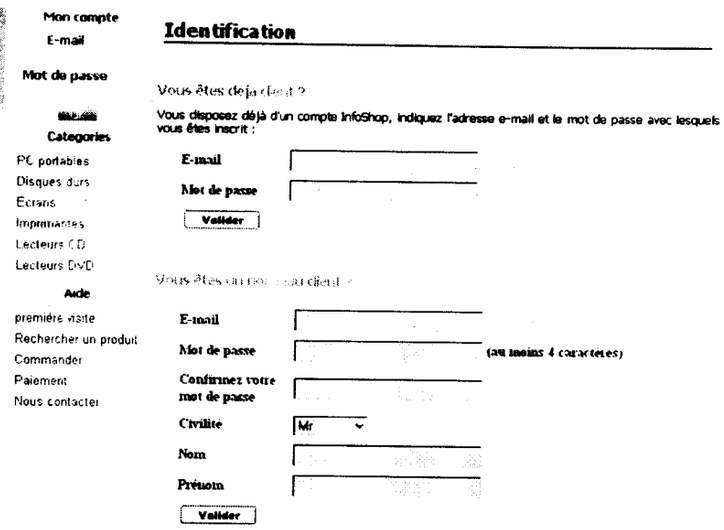


FIG. 7.8 – Identification.

L'étape suivante consiste à saisir les informations nécessaires à la livraison (figure 7.9) et au paiement (figure7.10).

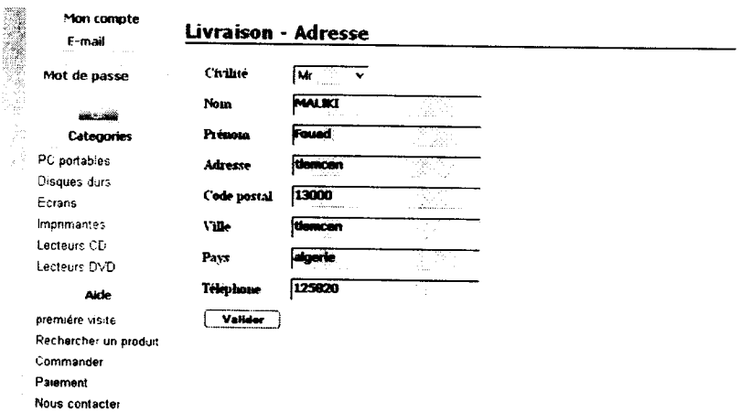


FIG. 7.9 – Adresse-Livraison.

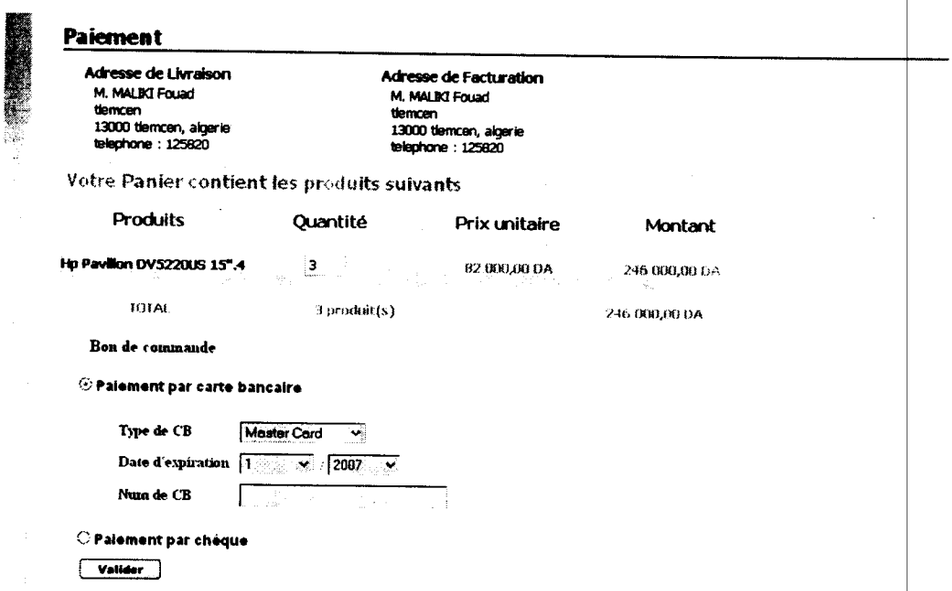


FIG. 7.10 – Paiement.

A la fin, un récapitulatif de la commande est affiché au client.

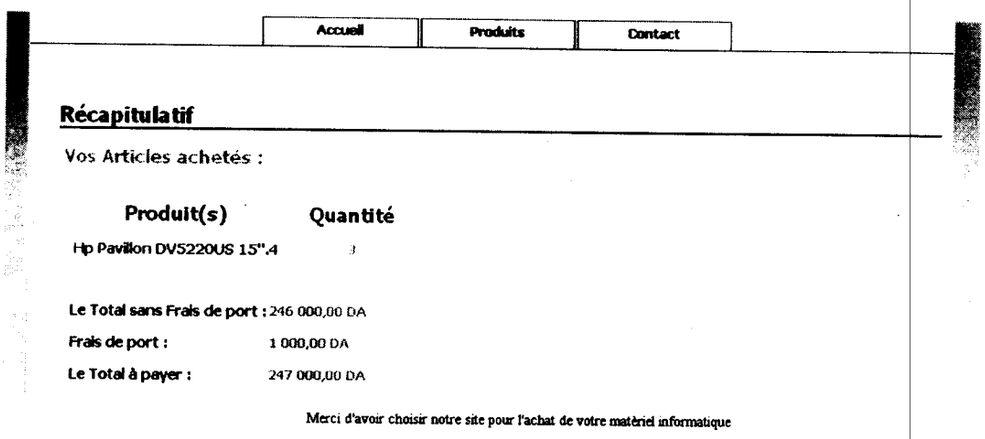


FIG. 7.11 – Récapitulatif.

Quand un client se connecte au site, il a la possibilité de consulter ses commandes, d'accéder à son panier et de modifier les informations de son compte.

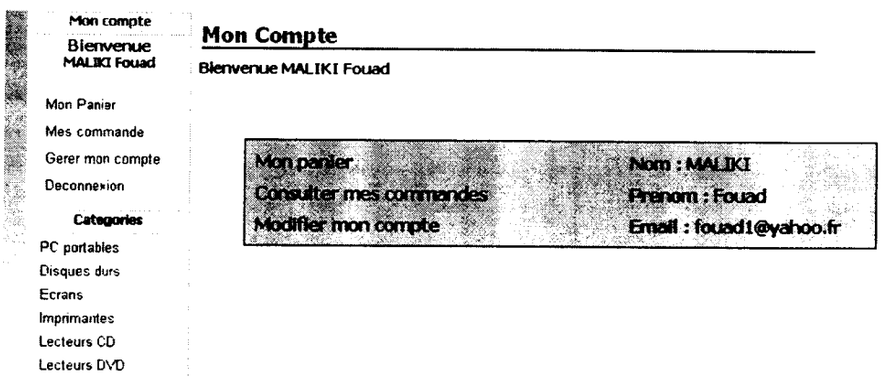


FIG. 7.12 – Gestion de compte.

Pour mieux guider l'internaute, il peut consulter la rubrique d'aide qui lui facilite la navigation dans le site.

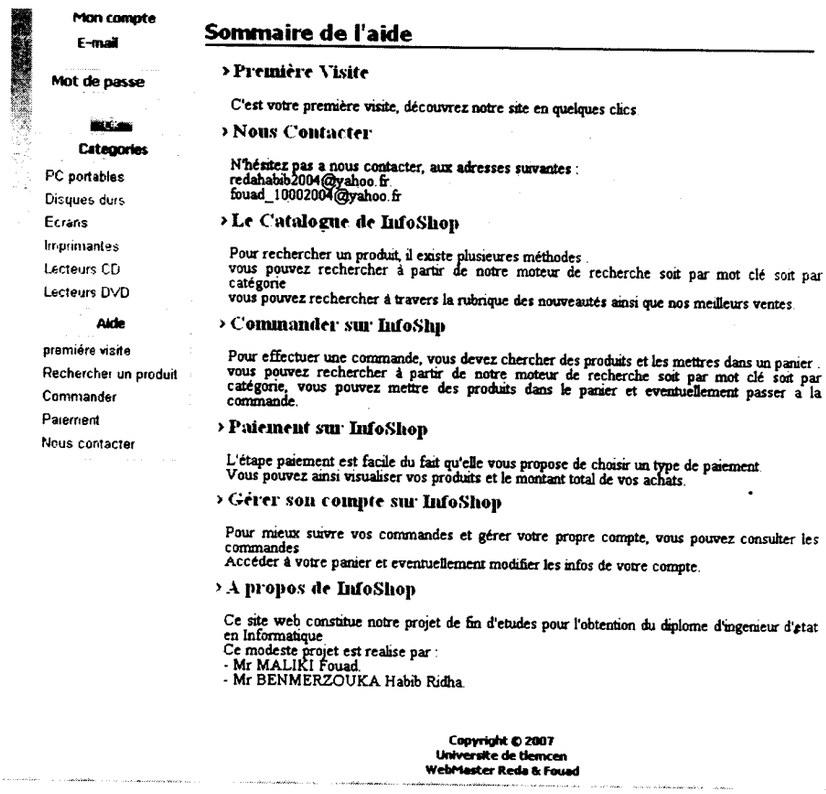


FIG. 7.13 – Aide.

7.3 Quelques exemples de code

Pour donner une idée générale sur la structure de notre application, voici en ce qui suit quelque partie du code source de la page index.jsp :

```
<%@include file="/header2.jsp"%>
<%@include file="/menugauche.jsp"%>
<%@include file="/menudroit.jsp"%>
<div id="main_content">
<%
 NouvFacadeHome home = null;
 NouvFacade remote = null;
 List l = new ArrayList();
 List le = new ArrayList();
 List lp= new ArrayList();
```

```
 Iterator it = null;
 Iterator it2 = null;
 List pc = new ArrayList();
 List ec = new ArrayList();
 List dis = new ArrayList();
 List imp = new ArrayList();
 List cd = new ArrayList();
 try {
   Object ref = context.lookup("NouvFacade");
   home = (NouvFacadeHome) PortableRemoteObject.narrow(ref, NouvFacadeHome.class);
   remote = home.create();
 } catch (NamingException ex) {}
   catch (CreateException ex) {}
l = remote.rechercherAll();
it = l.iterator();
while(it.hasNext()){
 le = (ArrayList)it.next();
 if(((String)le.get(5)).equals("PC Portables")) pc.add(le);
 else if(((String)le.get(5)).equals("Imprimantes")) imp.add(le);
  else if(((String)le.get(5)).equals("Disques Durs")) dis.add(le);
   else if(((String)le.get(5)).equals("Ecrans")) ec.add(le);
    else if(((String)le.get(5)).equals("CD") || ((String)le.get(5)).equals("DVD"))
                                                    cd.add(le);
  }
 out.print("<div class=\"bar\"><h1>Nouveautés</h1></div>");
%>
<div id="nouveaute">
  <div id="pcport">
   <div class="title">
     <h2>PC Portables</h2>
   </div>
   <table width="100%" align="center">
   <tr>
   <%
    it = pc.iterator();
while(it.hasNext()){
 lp = (ArrayList)(it.next());
 it2 = lp.iterator();
```

```
 int id = ((Integer)it2.next()).intValue();
 String img = (String)it2.next();
 String dispo = (String)lp.get(4);
 out.print("<td><ul><li><img src=\""+img+"\"></li>");
 out.print("<li>" + lp.get(2) + "</li>");
 out.print("<li><div class=\"prix\">" + prix(((Double)lp.get(3)).doubleValue())
                    + " DA </div></li>");
 out.print("<li>" + dispo + "</li>");
 if(dispo.equals("En stock"))out.print("<li><a href=\"panierserv2?cmd=ajout&id=
          "+id+"\"><img src=\"images/panier1.jpg\"></a></li></ul></td>");
}
  %>
  </tr>
  </table>
</div>
```

Nous donnons en ce qui suit quelques fragments de code de la page panier.jsp :

```
<%@include file="/header2.jsp"%>
<%@include file="/menugauche.jsp"%>

<div id="panier">
<%
     List li = (ArrayList)request.getAttribute("li");
     double total = ((Double)request.getAttribute("total")).doubleValue();
     String erreur = (String)request.getAttribute("erreur");
%>
</div>

<%
Iterator it = li.iterator();
int i=1;
String in ;
int q;
int nbqt = 0;
while(it.hasNext()){
  LignePanier l = (LignePanier)it.next();
  out.print("<div class=\"ligne\"><table width=\"100%\" border=\"0\"
  cellspacing=\"0\"><tr><td align=\"left\" width=\"30%\">");
```

```
  out.print("<a herf=\"index.jsp\">" +l.getNom() + "</a></td>");
  in="qt"+i;
  q=l.getQuantite();
  nbqt = nbqt + q;
  out.print("<td align=\"center\" width=\"20%\"><input type=\"text\"
            name=\""+ in +"\" value=\"" + q + "\"></td>");
  out.print("<td align=\"left\" width=\"20%\"><div class=\"prix\">" +
            prix(l.getPrix()) + " DA</div></td>");
  out.print("<td align=\"left\" width=\"20%\"><div class=\"prix\">" +
            prix(l.getPrixtotal()) + " DA</div></td>");
  out.print("<td align=\"left\" width=\"10%\"><a href=\"panierserv2?cmd=supp&id="-
          l.getIdprod()+ "\">supprimer</a></td></tr></table></div>");
  i++;
}
  %>
</div>
```

7.4 Conclusion

Tout au long de chapitre, nous avons essayé de donner une vision globale sur l'implémentation de notre application et cela par la présentation de quelques exemples de code.

Dans cette partie, nous avons présenté de façon détaillée le processus de développement depuis la phase de spécification jusqu'à l'implémentation.L'étude d'un tel type d'application nous a permis d'appréhender et de maîtriser la technologie EJB.

Conclusion Générale

A travers ce travail, nous avons présenté de façon sommaire et précise, une technologie importante dans le développement d'application distribuée.La technologie EJB définit par la plate-forme J2EE sert de support pour le développement d'application informatique puissante, performante et s'exécutant dans des architectures multi-niveaux.

De façon générale, la spécification EJB 2.1 décrit un modèle de composants développés en langage Java.Elle aborde en particulier les aspects suivants :

- rôles des différentes parties concernées par le développement, l'assemblage et le déploiement des composants ;
- interfaces entre client et conteneur et entre composant et conteneur ;
- trois catégories de composants (session, entity, message-driven) ;
- mécanismes de gestion des exceptions et des communications ;
- services sous-jacents (transaction,persistance,sécurité,...).

Les points forts de ce modéle sont les capacités de configuration (utilisation de descripteur de déploiement), d'intéropérabilité (via le ptocole RMI/IIOP) et de substutiabilité de ses composants (mécanisme de délégation entre classe d'inteface et classe d'implantation)[Ouss03].

Malgré tout ces points forts offerts par les EJB, ces derniers recélent quelques limites telles que :

- La prise en compte de la portabilité (Les interactions entre un composant et son support d'exécution ne sont pas toutes standardisées) ;
- L'indépendance aux langages d'implantation (pas de langage de description d'interface, utilisation limitée au langage de programmation Java).

La nouvelle version 3.0 de la spécification EJB apparue en fin 2006, définit une méthodologie de développement plus aisée que celle de la spécification 2.1, permettant ainsi de développer des applications performantes.

Actuellement, de nouvelles idées, de nouvelles perspectives, s'ouvrent sur le développement de plates-formes logicielles et de modèles de composants puissants dans le but d'accroître la qualité des systèmes logiciels.

Nous espérons que ce mémoire servira de guide aux futures promotions.

Bibliographie

[Lebl99] Rémi LEBLOND, *Vers une architecture N-Tiers*, Oral probatoire soutenu le 02/12/1999.

[CCM] www.commentçamarche.net.

[GeGrMe99] Jean-Marc Geib - Christophe Gransart - Philippe Merle, *CORBA : des concepts à la pratique*, 1999.

[Ouss03] Collectif, sous la direction de Mourad OUSSALAH, *Ingénierie des composants Concepts, techniques et outils*, Edition Vuibert 2003.

[Szyp98] Szyperski C., *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley 1998.

[Meye99] Meyer B., *On to component*, Rapport technique,IEEE Computer Janvier 1999.

[HaHe99] Harris J. et Henderson A., *A Better Mythotodologiyb for System Design*. Conférence d'interface homme-machine may 1999.

[Durr03] Laureant Durriel, *Developpement Java/J2EE avec JBuilder*, Eyrolles 2003.

[RoSrBr05] Ed Roman, Rima Patel Sriganesh et Gerald Brose, *Mastering Entreprise Java Beans*, Wiley 2005.

[Ecke00] Bruce Eckel, *Penser en Java 2nd Edition Traduction en français de « Thinking in Java » de Bruce Eckel http ://www.eckelobjects.com/* 2000.

[RoVa04] Pascal Rouques, Franck Vallée, *UML 2 en action De l'analyse des besoins à la conception J2EE*, Eyrolles 2004.

[Roqu03] Pascal Roques, it Les Cahiers du programmeur UML Modéliser un site e-commerce, Eyrolles 2003.

[BeAh06] Benamar a. et Ahmet h., *Développement d'une application distribuée par la norme CORBA*, mem ing en informatique univ Abou Bekr Belkaid Tlemcen 2006.

[Audi06] Laurent AUDIBERT, *UML 2.0*, IUT de Villetaneuse 2006.

[Fowl04] Martin FOWLER, *UML Distilled Third Edition*, Addison-Wesley 2004.