MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI, TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET DE L'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

Mémoire de fin d'études

Présenté en vue de l'obtention

du Diplôme de Master en Informatique

Thème:

Conception et réalisation d'un « module de facturation» pour un ERP de transport.

Dirigé par :

M^me^ TAOURI Dalila.

M^r^ SAHNOUNE Nabil.

M^r^ LADRAA Karim.

Présenté par :

M ^elle^ NESNAS Radia.

M^r^ SI RAMDANE Noureddine.

Promotion 2012

Remerciements

Á travers ce modeste travail, nous tenons à remercier vivement notre promotrice Mme TAOURI Dalila, Mr SAHNOUNE Med Nabil et Mr LADRAA Karim qui nous ont honoré en acceptant de nous encadrer et pour leurs précieux conseils et orientations, ainsi que la confiance qu'ils ont placé en nous tout au long de la préparation de ce projet.

Nos remerciements et gratitudes s'adressent aussi aux messieurs le président et les membres de jury d'avoir accepté d'examiner notre travail.

Nos vifs remerciements vont à l'ensemble du personnel de Marine Soft spécialement Mme AMRANI Asmahane, Melle JAHNIT Hadjer, Mr AMANI Sief, Mr AIT GUERBI Kamel.

Nous remercions enfin tous ceux qui ont contribué de près ou de loin à l'élaboration de ce travail.

Dédicaces

Je dédie ce travail :

A celle et à celui qui m'abreuvent d'amour et d'affection intarissable, sources de mon bonheur et ma raison d'être, ma très chère mère et mon très cher père, qui ont su guider mes pas vers ce que je suis devenu aujourd'hui. Que Dieu me les garde encore aussi longtemps que je vivrai.

A la mémoire de mon grand père paternel «Si Med Larbi ». Que dieu le garde parmi ces bien aimés.

A ma grande mère paternelle « Ouerida » ainsi mon grand père maternel « Ali ».

A mes adorables frères : Khaled & Ismail ainsi mes adorables sœurs : Lynda, Hanane & Sabah (Soraya).

A mes tantes, mes oncles, cousins et cousines.

A ma collègue « Radia » qui, par sa participation effective a permis à ce projet de voir le jour ainsi que toute sa famille.

A la très chère « Zohra » que je remercie beaucoup pour tout l'aide qu'elle m'a apporté.

A tout mes amis(es) que j'aime beaucoup, surtout Farid, Idir, Lyly, Kahina, Hayat, Wisseme et Yacine.

A toute l'équipe de Marine Soft ainsi toute personne de la promotion 2011/2012.

A toute personne qui a contribué de prés ou de loin à la réalisation de ce projet.

SI RAMDANE Noureddine.

Dédicaces

Je dédie cet humble travail :

A mes chers parents qui m'ont toujours soutenu par leur indéfectible amour, leur patience et leur réconfort.

A la mémoire de ma grand mère paternelle Ouerdia que dieu la reçoit dans son vaste paradis.

A ma grand mère maternelle « Ouiza » qui j'espère sera présente à toutes mes joies.

A ma sœur Salima et à son mari Achour.

A ma sœur Ouerdia et son fiancé Farid.

A mon petit frère Said que j'adore.

A toute ma famille qui est toujours là pour moi.

A mon binôme et sa famille.

A tous mes amis(es) sans eux je n'arriverai pas à avoir tout le courage et toute la volonté de résister aux moments durs.

A toute l'équipe de Marine Soft ainsi toute personne de la promotion 2011/2012.

A toutes les personnes qui m'aiment, à ceux qui ne sont pas ici avec moi mais qui sont toujours dans mon cœur.

NESNAS Radia.

Sommaire

Partie1

Chapitre I

Chapitre II

Partie2

Chapitre 1

Chapitre II

Chapitre II

Liste des figures

Partie 1

Chapitre I

Chapitre 2

Partie 2

Chapitre 1

Partie 3

Chapitre 2

Liste des tableaux

Partie 1

Chapitre 1

Chapitre 2

Partie 2

Chapitre 1

Partie 3

Chapitre 1

Motivations et objectifs

En vu de l'obtention du diplôme Master 2 en informatique, et pour compléter notre cursus universitaire par une expérience professionnelle, il nous a été proposé d'effectuer un stage de fin d'études par une entreprise appelée **Marine Soft,** société d'ingénierie en informatique, sise à Alger (voir chapitre I de la partie 1).

De nos jours, toute entreprise est prête à investir des sommes considérables dans l'implantation des technologies logicielles afin d'améliorer ses services, accroître son agilité et sa flexibilité, réduire ses coûts, augmenter sa production et faire face aux défis du marché. En effet, vu la croissance des activités au sein des entreprises, de gérer efficacement toutes ces fonctions s'avère de plus en plus complexe et difficile.

Pour surpasser ces difficultés, une entreprise doit utiliser des outils optimisés et adaptés, facilitant les tâches et offrant des fonctionnalités riches et utiles. Parmi ces outils, nous trouvons les systèmes intégrés de gestion tels que les ERP (Entreprise Ressources Planning).

Les ERP sont des outils de gestion et d'analyse permettant d'optimiser la diffusion des informations en interne, d'améliorer les processus de gestion et d'automatiser les tâches. Ceci augmente énormément la réactivité des entreprises et leurs agilités.

C'est dans ce contexte que s'intègre notre stage d'immersion en entreprise qui a pour objectif de concevoir et de réaliser un Système d'Information, SI, « Gestion de facturation » pour un ERP de transport.

Cette application doit prendre en charge les fonctions suivantes :

1. Facture pro format, initiale, avoir et complémentaire.
2. Un outil de recherche multi critères des factures et des règlements.
3. Le suivi des règlements des factures à crédit.
4. Passerelle vers le logiciel de comptabilité.
5. Une couche de communication avec les autres modules de l'ERP.
6. L'IHM de l'application doit être entièrement paramétrable.
7. Multilingue.

Organisation du mémoire

Pour mener à bien notre travail, nous avons organisé notre mémoire en trois parties:

- **1ere partie : Contexte de mémoire**

  Qui est composé des chapitres suivants :

  - ✓ **Présentation de l'organisme d'accueil :** en l'occurrence, Marine Soft ainsi que ses différents produits.
  - ✓ **Analyse et étude de l'existant :** analyse, étude et critiques des applications principales développées par Marine Soft.
  - ✓ **Proposition de la solution :** suggestion de la solution de l'ERP GLS (Global Logistic Software) contenant le noyau facturation.

- **2eme partie : Conception**

  Présentant les chapitres suivants :

  - ✓ **Analyse et conception :** Identification des besoins et des acteurs de l'application et proposition d'une conception en utilisant le langage de modélisation UML.

- **3eme partie : Réalisation**

  Englobe les chapitres suivants :

  - ✓ **architectures et outils technologiques :** présente l'ensemble des outils et logiciels, ainsi que la plate-forme et les langages de programmation Web utilisés pour la mise en place de l'application.
  - ✓ **Architecture de l'application :** présente le schéma applicatif de notre application, son principe de fonctionnement ainsi que quelques interfaces de notre plate- forme.

Et pour terminer, une conclusion synthétise notre travail en mettant en relief les apprenants de ce dernier et ouvrant des perspectives pour d'éventuelles améliorations.

Démarche de réalisation du projet

Il est important de bien faire la distinction entre une méthode qui est une démarche d'organisation et de conception en vue de résoudre un problème informatique, et le formalisme dont on peut utiliser dans le but d'exprimer le résultat voulu. Les grandes entreprises ont souvent leurs propres méthodes de conception ou de réalisation de projets informatiques. Il n'était donc pas réaliste de tenter de standardiser une méthodologie de conception.

1. Démarche

La démarche adoptée au cours du processus de mise en œuvre de ce projet, se résume en neuf étapes. Voici la figure qui les illustre :

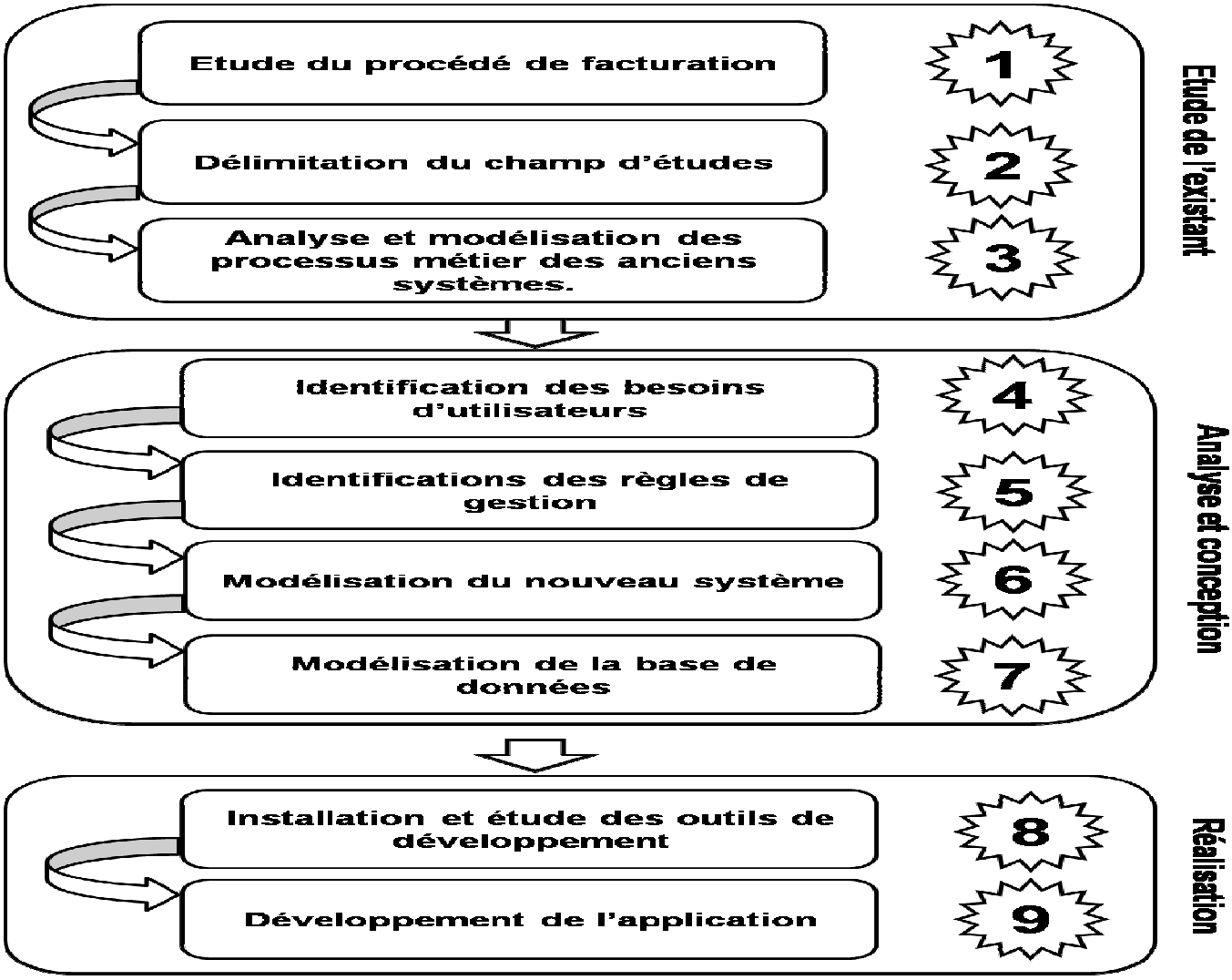


Figure 1 : Démarche de réalisation du projet

l'équipe de développement de Marine Soft, d'une formation en « Management de projet ». M[r] Philipe LUTMAN, expert en la matière a été sollicité par l'entreprise pour assurer cette formation d'une durée de trois jours.

Nous avons, entre autres, appris, au cours de cette formation, à faire le découpage, l'estimation des charges et la planification dans le cadre d'un projet.

L'entreprise avait besoin de faire une estimation des charges dans le cadre du projet ERP GLS, mais pour cela il fallait écrire d'abord le cahier des spécifications fonctionnelles détaillées de la totalité des modules constituant l'ERP. L'entreprise nous a donc demandé de rédiger les spécifications fonctionnelles détaillées et de tracer un planning concernant la réalisation de notre projet.

Cette initiative, de la part de l'entreprise, avait pour objectif de nous pousser à la réflexion, traiter et spécifier les moindres détails. Donc avoir une très bonne métrise des fonctionnalités du nouveau système, en rédigeant un cahier des spécifications fonctionnelles détaillées.

La rédaction de ce cahier, a duré deux mois, durant lesquels l'expert et les responsables de Marine Soft nous ont encadrés. On a abouti à la quatrième version qui est validée.

Ci-dessous le planning du noyau « **Facturation** » :

Etape	**Avril**				**Mai**				**Juin**				**Juillet**				**Août**				**Septembre**			
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								

Tableau: Planning de la réalisation du projet.

Partie I

Contexte du mémoire

Chapitre I : Présentation de l'organisme d'accueil.

- Organisme d'accueil.
- Problématique et perspectives de développement.
- Conclusion.

Chapitre II : Analyse et études de l'existant

- Présentation du procédé de facturation
- Choix d'approche.
- Diagnostic du système informatique existant.
- Bilan du diagnostic (anomalies et causes).
- Présentation de la solution informatique.

1. Organisme d'accueil

1.1. Historique et présentation

MARINE SOFT Sarl, EDITEUR ET INTEGRATEUR DE SOLUTIONS LOGICIELLES est une Société de services et d'ingénierie en informatique, entièrement créée en 1998, spécialisée dans la conception, la réalisation, la mise en route et l'assistance technique de divers logiciels relatifs à des activités liées au transport.

Le conseil et l'audit en info-logistique positionnent MARINE SOFT comme un véritable partenaire de développement pour les professionnels de la chaîne de transport. Elle relie les entreprises privées, leurs organisations, les administrations et les institutions publiques pour former une chaîne logistique optimisée en adoptant des processus communs. Ses applications et ses solutions sont spécifiquement conçues et sont entièrement dédiées aux besoins des professionnels de la logistique du transport.

La force de ses produits vient de leur haute qualité opérationnelle qui est le résultat d'un travail minutieux, conduit dans des conditions opérationnelles réelles dans les différentes étapes depuis la conception jusqu'à la mise en exploitation.

1.2. Missions et objectifs de Marine Soft

Marine Soft Sarl a pour mission la conception, la réalisation, la mise en route et l'assistance technique de divers logiciels adaptés à l'environnement institutionnel et économique algérien et international.

Les principaux objectifs de Marine Soft sont :

- Anticiper les réels besoins des entreprises en matière de logiciels et proposer des produits pertinents, performants, faciles à mettre en œuvre, souples d'utilisation, extrêmement fiables en exploitation, surtout ouverts et évolutifs.
- Privilégier l'essentiel, à savoir l'utilisateur et l'opérationnel, avec une priorité absolue pour la qualité de service et la satisfaction totale des clients.
- Faire le suivi dans le temps, afin de prendre en compte les travaux et les recommandations des instances normatives, l'environnement institutionnel, le contexte économique national et international, les communautés sectorielles et les donneurs d'ordres;
- Travailler le plus possible en partenariat avec les principaux opérateurs économiques d'une part et les éditeurs de progiciels de gestion d'autre part.
- Conseil et Audit en info logistique.

1.3. L'organigramme de Marine Soft

La représentation graphique de la structure fonctionnelle et de l'organisation hiérarchique des services de Marine Soft sont comme suit :

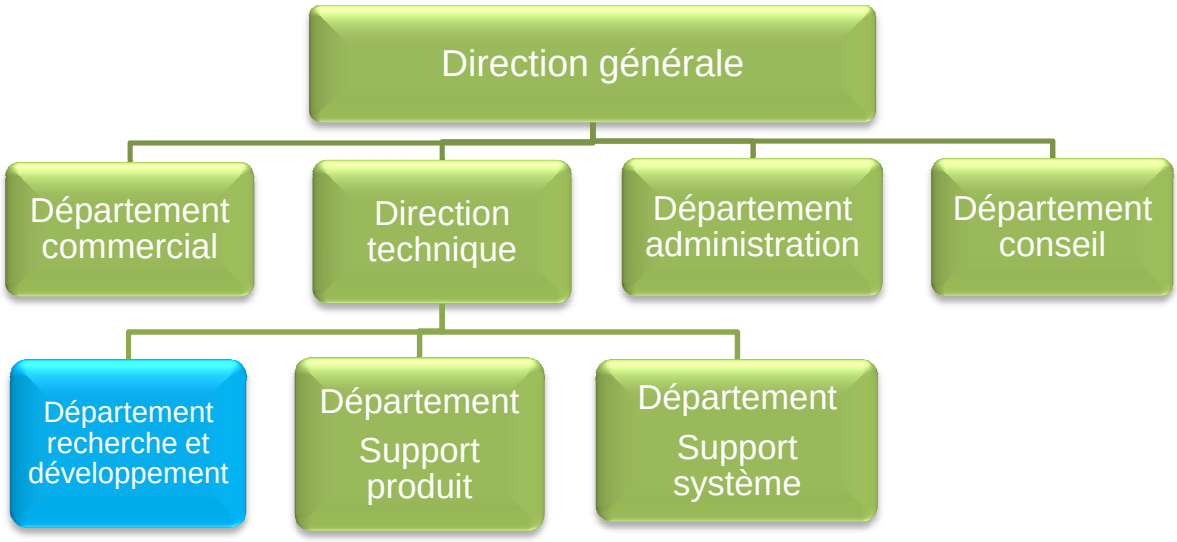


Figure I.1 : Organigramme de Marine Soft

1.4. Description de l'organigramme de Marine Soft

Le tableau ci-dessous présente les missions de chaque département de cette entreprise :

Département	Missions
Département Recherche et Développement	-Développement de la gamme de progiciels de MARINE SOFT. -Maintenance progiciels. -Assistance technique.
Département Support Produit	-Formation des utilisateurs. -Installation et paramétrage des logiciels. -Assistance téléphonique. -Télémaintenance. -Service help (outils d'aide au support).
Département support Système	-Gestion des parcs informatiques. -Assistance téléphonique et télémaintenance. -Maintenance (hardware, administration, serveur). -Configuration de messagerie. -Suivi des achats du matériel.
Département administration	-Comptabilité/paie. -Facturation/Recouvrement. -Administration des ventes (Contrats &Conventions de Formation).

Département conseil	-Conseil (architecture des systèmes informatiques, études comparatives, étude de migration, optimisation).

Tableau I.1 : Description de l'organigramme de Marine Soft.

1.5. La clientèle de Marine Soft

Marine soft développe ses produits pour des organismes qui ont des activités liées au domaine maritime comme :

- EPAL [Entreprise Portuaire d'Alger].
- AMS [Algerian Maritime Services](EUKOR, Sloman Neptun, Cargo Levant).
- Arkas Algérie (Groupe ARKAS).
- Mondial Shipping Company (Marfret, UASC).
- TIBA (Evergreen Marine Corp).
- Cargo Levant.
- Sloman Neptun.
- Groupe d'Alessandro Tunisie (Groupe Grimaldi).
- Tunisamar Tunisie [Groupe Ismar] (Hanjin Shipping).
- MILAHA SERVICES (Brointermed Lines Limited).
- Rail Transit (Groupe Société Nationale des Transports Ferroviaires).
- ALTERCO [ALgerian TeRminal COntainer] (Groupe CMA-CGM).
- ATERCO (Groupe ARKAS).
- ACS [Algerian Containers Services] (filiale Entreprise Portuaire d'Alger).
- Sudcargo Algérie (finale Sudcargos).
- Satrans [Services annexes aux transports maritimes].

1.6. Les produits de Marine soft

1.6.1. Les produits développés

Voici les applications réalisées par l'équipe de développement de Marine Soft :

Désignation	Date de création	Composants	Architecture	Utilisation
GAM Global Agency Management	1998	-Le suivi des Opérations Navire. -La documentation. -Déclaration douanière. -Suivi des situations BL. -Gestion des Conteneurs.	Client/serveur **2tiers**	Agences consignataires

		-Etablissement des comptes escale et des comptes armateur. -**Facturation.**		
ZSD **Zone Sous Douane**	1999	-Suivi de la documentation. -La gestion de la zone. -**Facturation.**	Client/serveur **2tiers**	Le port d'Alger Les ports secs.
MSGMAN **Gestion de la manutention**	2002	-Suivi des Opérations Navire. -Gestion des moyens humains et matériels. -Gestion de l'exploitation. -Suivi de l'état de débarquement & embarquement. -Statistiques. -**Facturations.**	Client/serveur **2tiers**	Le port d'Alger Les ports secs
MS TRANSIT	2001	-Suivi des dossiers. -Déclaration. -La gestion du crédit. -Gestion des Conteneurs. -Les réimprimé. -Suivi de la tarification. -Établissement des comptes clients. **-Facturation.**	Client/serveur **2tiers**	Agences transitaires
MS Transport	2002	-La gestion des transferts. -La gestion des transports. **-Facturation.**	Client/serveur **2tiers**	Agences transitaire.
GPP **Gestion des Parcs Pleins**	2005	-Gestion des Conteneurs. -Suivi des différents déplacements du conteneur. **-Facturation.**	Client/serveur **2tiers**	Parc de conteneurs pleins

GPV Gestion des Parcs Vides	2006	-Gestion des Conteneurs. -Suivi des différents déplacements du conteneur. **-Facturation.**	Client/serveur **2tiers**	Parc de conteneurs vides
GRC Gestion de la Réparation Conteneurs	2006	-Le suivi des mouvements conteneurs (entrée /sotie). -L'établissement des états conteneur **-Facturation**	Client/serveur **2tiers**	Parc qui s'occupe de la réparation des conteneurs endommagés.

Tableau I.2 : Les produits développés de Marine Soft.

1.6.2. Le produit en cours de développement

L'équipe Marine Soft développe une nouvelle application qui est présentée dans le tableau ci-dessous:

La désignation	Type	Composant	Langage de programmation	Architecture	Utilisateur
GMAO Gestion Maintenance Assistée par Ordinateur	Progiciel	-Gestion patrimoine. -Gestion stock. -Gestion d'intervention. -Gestion d'achat. -Gestion de ressources humaines.	JAVA	Client/serveur 3tiers	Port d'alger

Tableau I.3: Le produit en cours de développement de Marine Soft.

GMAO (Gestion de Maintenance Assistée par Ordinateur) est un progiciel qui permet d'automatiser la gestion de la maintenance dans une entreprise dans différents secteurs tel que la production, l'exploitation ou l'administration.

Remarque : *On remarque bien que tous les produits développés (Voir Tableau I.2) ont une architecture de type client/serveur 2tiers, par contre le produit en cours de développement (Voir Tableau I.3) a une architecture de type client/serveur 3tiers.*

2. Problématique et perspectives de développement

2.1. Problématique

Le problème qui se pose au niveau des applications de Marine Soft réside :

1) Dans l'architecture de type client/serveur 2tiers ne fonctionnant que sur WINDOWS.

Les produits de notre organisme d'accueil ne répondent plus aux attentes de leurs clientèles d'un point de vue technologique : solutions multi-plateformes, interface Web technologique basée sur le n-tiers ...etc.

2) Dans la facturation qui est standard et identique pour tous les produits logiciels de Marine Soft, mais cette dernière est dupliquée dans tous les produits logiciels, ce qui engendre une maintenance lourde et une mise à niveau coûteuse.

2.2. Perspectives de développement

a. De l'architecture client/serveur 2 tiers vers l'architecture client/serveur 3 tiers

Comme cité précédemment, le concept « **2 Tiers** » laisse place à celui du « **3Tiers** ». Cette transition n'est pas le fruit du hasard, mais une démarche réfléchie et entreprise suite à un premier diagnostic établi par la direction.

b. Vers un module de facturation standard

b1. Développement de l'ERP GLS

Aujourd'hui Marine soft a un nouvel objectif qui est la réalisation de l'ERP GLS.

GLS est un **ERP** («Enterprise Resource Planning»), aussi appelé **« Progiciel de Gestion Intégré »** (PGI).*Un ERP est un progiciel qui intègre les principales composantes fonctionnelles de l'entreprise*[1]*.* A l'aide de ce système unifié, les utilisateurs de différents métiers travaillent dans un environnement applicatif identique qui repose sur une base de données unique. Ce modèle permet d'assurer l'intégrité des données, la non-redondance de l'information, ainsi que la réduction des temps de traitement ».

[1] Livre : Piloter un projet ERP, par Gean-Luc DEIXONNE, 2011.

L'ERP GLS est destiné à la logistique de transport de marchandise, qui contient plusieurs modules que nous illustrons dans le schéma ci-dessous :

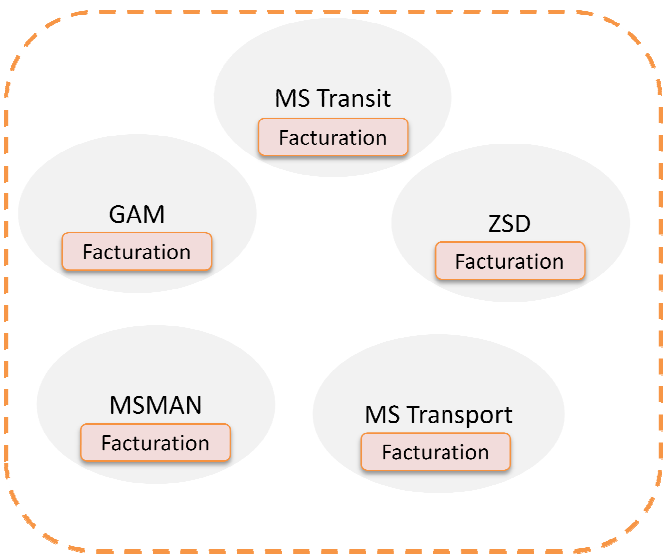


Figure I.1 : Système actuel de Marine Soft : Intégration du module facturation dans chaque application (duplication).

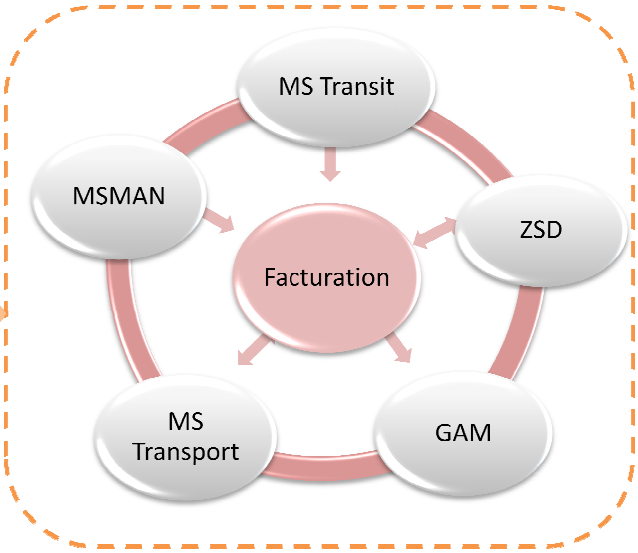


Figure I.2 : Architecture de **GLS**: Unification du module facturation (Futur système).

2.3. Organisation de l'équipe de recherche et développement

Le projet GLS a été confié à une équipe de développement constituée des ressources suivantes :

- Un expert d'IBM.
- Le directeur général.
- Le directeur technique.
- Cinq ingénieurs de développement.
- Trois techniciens support produit.
- Sept stagiaires répartis en trois binômes dont le nôtre et un monôme.

La répartition des tâches fut simple, de sorte qu'un développeur encadre et suit de près ses stagiaires. Chaque couple développeur/stagiaire se voit affecter la responsabilité de mener à bien la conception, la réalisation et la mise en œuvre de l'application qui lui a été confiée. Les directeurs font le suivi de la totalité du projet, c'est-à-dire de toutes les applications de GLS. L'expert, un spécialiste des nouvelles technologies de développement assurera le choix technologique des outils, le choix des architectures et les Framework qui seront utilisés pour la réalisation de nos projets.

Dans le cadre de ce projet, Marine Soft nous a confié la réalisation du noyau « **Facturation** » et la mise en place du système de la communication avec les autres applications de GLS.

Conclusion

Dans ce présent chapitre, nous avons présenté l'organisme d'accueil **« Marine Soft »,** en citant ses missions et ses objectifs. Nous avons vu également la structure fonctionnelle et l'organisation hiérarchique de tous les départements de l'entreprise ainsi que ses différents clients.

Nous avons fait ensuite, le tour de toutes les applications existantes (les produits développés) et celles en cours de développement, ainsi que les perspectives de développement.

Nous présenterons dans un premier temps le procédé de facturation ainsi que le choix de l'approche processus. Nous effectuerons ensuite le diagnostic du système informatique existant qui prendra en charge (entre autres) la facturation. Nous terminerons par dresser la liste des critiques, ainsi qu'un schéma global de la future solution informatique autrement du futur système automatisé (futur ERP).

1. Présentation du procédé de facturation

La facturation est basée sur un enchainement de processus à savoir :

- Facture pro format.
- Facture initiale.
- Facture avoir/complémentaire.
- Suivi règlement des factures à crédit.
- Recherche multi critères.
- Edition.

1.1. Les processus liés à la facturation

a. La facture pro format

Une facture pro format est une simulation de la facture. C'est une facture qui n'a pas de valeur comptable. Le fournisseur s'y engage sur la base de conditions à fournir la marchandise commandée. Le client par contre ne s'engage que lorsqu'il passe la commande ferme.

Remarque

La facture pro format n'a pas de numéro.

b. La facture initiale

La facture initiale est un document comptable représentant la preuve d'un achat ou d'une vente de produits ou de services. Le fournisseur est tenu de délivrer la facture dès la réalisation de la vente ou la prestation du service. L'acheteur doit la réclamer.

c. La facture avoir/complémentaire

Il est possible qu'une entreprise soit amenée à modifier sa facture Initiale qu'elle a déjà validé, et ceci dans deux cas :

- **Facture avoir**

Si l'acheteur a constaté qu'il est victime d'une surfacturation. Il réclamera alors la différence au vendeur qui la lui réglera sous forme d'une facture négative appelée « facture Avoir ». C'est un contrat d'erreur en défaveur du vendeur-fournisseur, donc en faveur du client-acheteur.

- **Facture complémentaire :**

Si le vendeur a sous-facturé les prestations ou produits vendus, le fournisseur sera rétabli dans ses droits en établissant une facture positive appelée « facture Complémentaire ». C'est un contrat d'erreur en faveur du vendeur-fournisseur, donc en défaveur du client-acheteur.

Les factures Avoir et Complémentaire n'existent pas si la facture initiale n'a pas été établie.

d. Suivi règlement des factures à crédit

Permet de régler les factures à crédit (initiale ou complémentaire) des clients conventionnés. Une facture à crédit est une facture non réglée lors de sa création.

Remarque :

L'ancien système informatique implémente l'ensemble de ce procédé avec intégration de deux fonctionnalités qui sont :

1. Recherche multi critères

Elle permet de rechercher les factures (initiales, avoirs et complémentaires) et les règlements selon différents critères de recherche qui peuvent être combinés, qui sont :

- Le numéro de facture ou du règlement.
- La date de facturation ou du règlement.
- Le nom de client.
- Etc.

2. Edition

Elle permet l'édition des factures, l'historique de la facture et les bons de reçu. On retrouve dans la plupart des fonctions, l'édition. Au lieu de les détailler dans chaque partie, on a préféré la considérer comme étant une fonction, qu'on détaillera à part.

1.2. Enchainement des processus liés à la facturation

Ce diagramme résume l'enchainement des processus liés à la facturation.

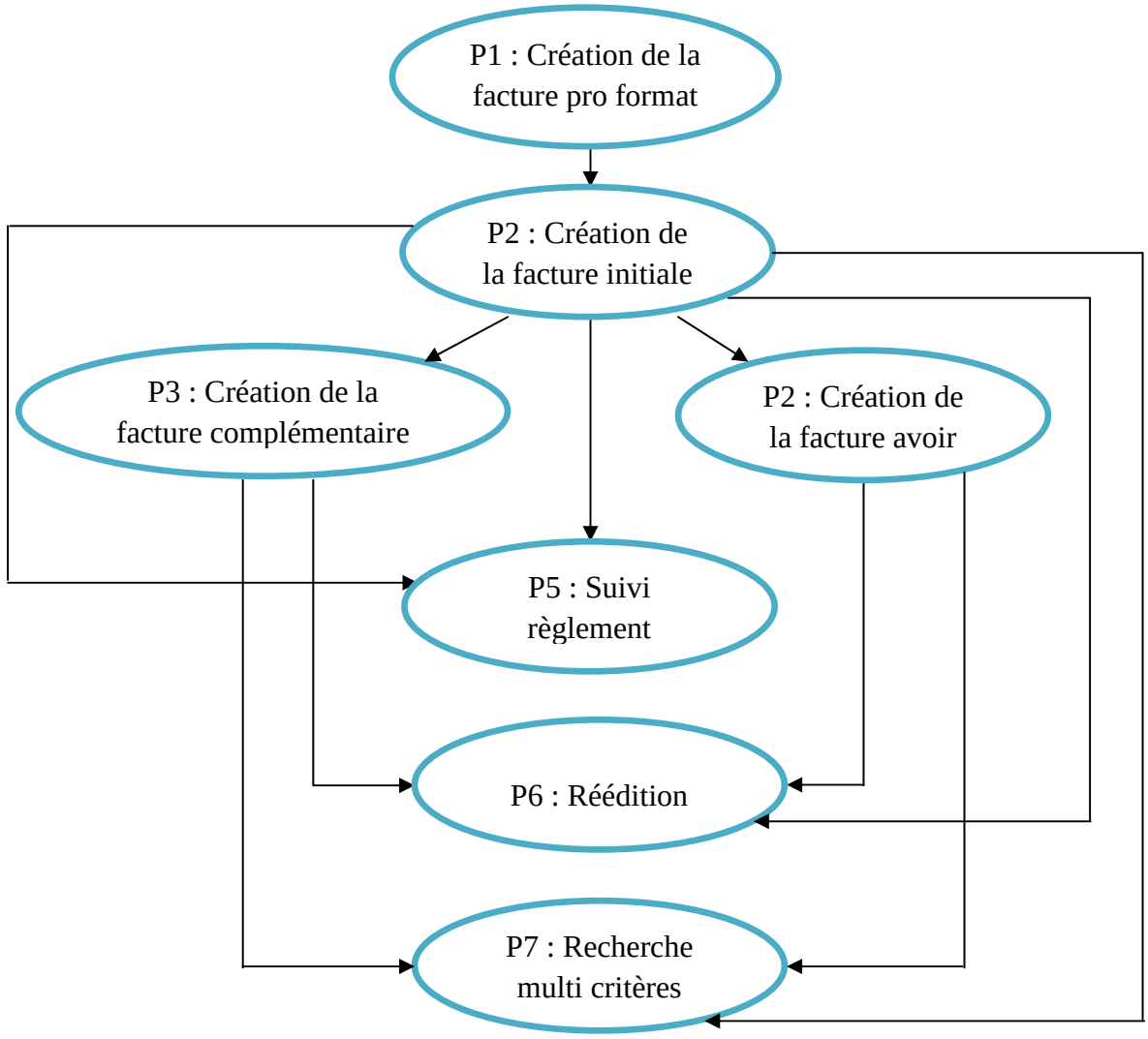


Figure II.1: Diagramme d'enchainement des processus liés à la facturation.

2. Choix d'approche

Adoptée par l'entreprise en début d'année, la norme **ISO 9001** fait partie de la série des normes ISO 9000, relatives aux systèmes de gestion de la qualité. Elle définit des exigences concernant l'organisation d'un système de gestion de la qualité. En tant que liste d'exigences, elle sert de base à la certification de conformité de l'organisme. La version en vigueur de ISO 9001 est la version datée de 2008. Les exigences y sont relatives à quatre grands domaines :

1. Responsabilité de la direction
2. Système qualité

3. Amélioration continue
4. Processus

Cette norme encourage l'adoption de l'approche processus lors du développement, de la mise en œuvre et de l'amélioration de l'efficacité d'un système de management de la qualité, afin d'accroître la satisfaction des clients par le respect de leurs exigences.

2.1. L'approche processus[1]

L'approche processus est une méthode d'analyse et de modélisation. Elle Consiste en l'identification et management méthodiques des processus utilisés dans un organisme, et plus particulièrement les interactions de ces processus. [ISO 9000].

2.2. Concepts de base

2.2.1. Processus

Ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie. [ISO 9000].

2.2.2. Sous-processus

Un sous-processus est une étape du processus. Il s'agit d'une suite chronologique de tâches réalisées par des personnes ou des applications informatiques différentes à des moments et en des lieux différents. Cet ensemble de tâche a un début et une fin pour obtenir un résultat intermédiaire.

2.2.3. Processus élémentaires

Ensemble de sous processus liés, qui dépendent directement ou indirectement des mêmes éléments d'entrée et produisant directement ou indirectement le même élément de sortie.

Remarque

Les types de processus conforment à la liste de normes ISO 9000, sont au nombre de trois à savoir: processus de réalisation, processus de support et processus de management.

2.2.4. Les processus de réalisation (opérationnels)

Ensemble de processus allant du client au client. Ils permettent la réalisation du produit ou du service fourni par l'entreprise à ses clients et correspondent ainsi à l'activité «Métier» de l'organisation. Ces processus couvrent le cycle de vie du produit (service) et ont évidement un impact direct sur la satisfaction du client.

[1] Livre : management de la qualité management de processus, édité et diffusé par l'Association Française de Normalisation (AFNOR), en juin 2000.

2.2.5. Les processus de support (soutien)

Ensemble des processus donnant les ressources aux autres processus. Ils représentent une activité interne, généralement transversale, permettant d'assurer le bon fonctionnement de l'entreprise. Ils contribuent au succès des processus de réalisation, en leur fournissant les moyens(ressources humaines, infrastructures, environnement de travail et information) relatifs à un bon déroulement.

2.2.6. Les processus de management (pilotage)

Correspondent à la détermination d'une politique et d'une stratégie pour l'organisation et au pilotage des actions mises en œuvre pour atteindre ses objectifs. Ces processus sont sous la responsabilité de l'équipe dirigeante et agissent directement sur le fonctionnement de l'organisme et sur sa dynamique d'amélioration. Ils assurent la cohérence des processus de réalisation et support.

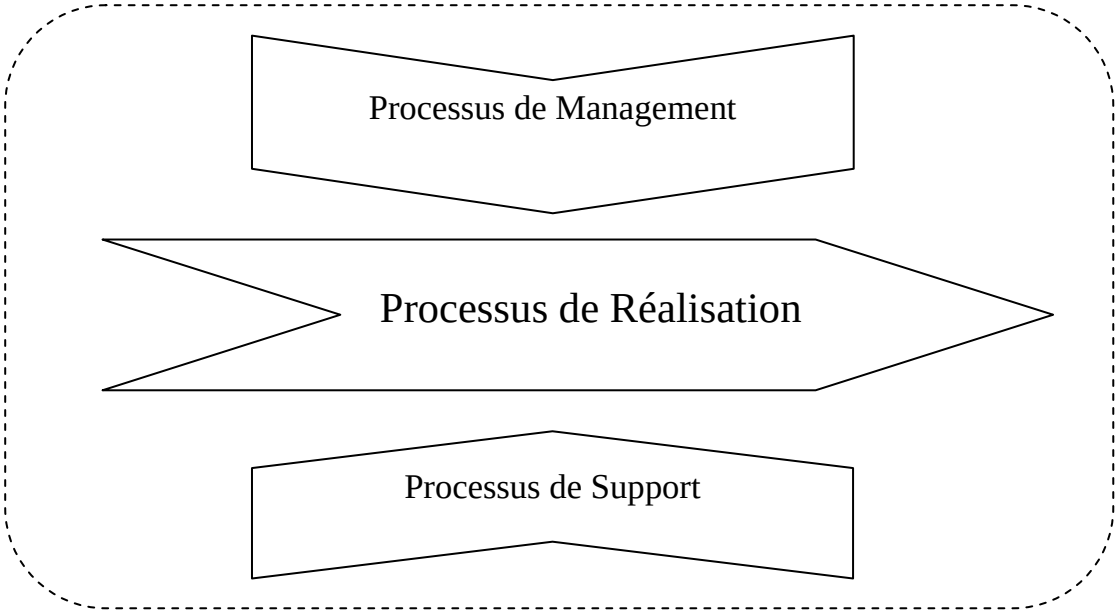


Figure II.2 : Disposition des processus dans un contexte de management qualité.

2.3. Restriction

Contrairement au processus de réalisation, les processus de support et de management sortent du cadre de cette étude. C'est pour cette raison qu'il ne sera fait aucune référence à ces deux type de processus, au cours du reste de ce chapitre.

2.4. Représentation normalisé d'un processus

Un processus est caractérisé par :

- ✓ Un *Non.*
- ✓ Un ou plusieurs *éléments d'entrée* (EE) : ressources nécessaires pour démarrer un processus. Ils proviennent d'un ou plusieurs fournisseurs externes ou internes (autres processus).
- ✓ Un ou plusieurs *éléments de sortie* (ES) : résultat de la transformation des éléments d'entrée par le processus. Ils sont destinés à un ou plusieurs clients externes ou internes (autres processus).
- ✓ Suite d'activité (processus élémentaires PE) qui transforme les éléments d'entrée en éléments de sortie en apportant une valeur ajoutée (VA).

Voici la représentation graphique d'un processus via l'notation qui sera utilisée au cours de cette étude chapitre.

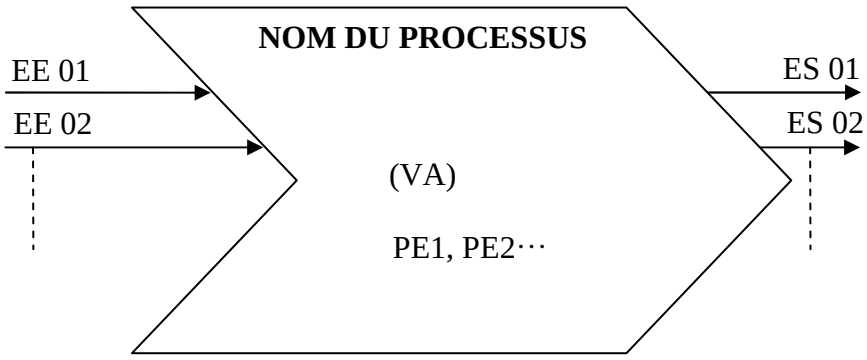


Figure II.3: Représentation normalisé d'un processus.

2.5. Représentation normalisé d'un ensemble de processus de réalisation

La phase d'identification des processus doit avoir un résultat concret, clair, facilement partageable, compris par un grand nombre d'acteurs, donc, qu'il est possible de soumettre à validation. Une simple liste commentée peut, théoriquement, suffire à exprimer le résultat de cette approche, mais une représentation schématique, simplificatrice et mémorisable convient mieux. On parle alors d'élaborer une cartographie des processus.

La *figure II.4* est un exemple de ce que doit être en principe, une cartographie de processus de réalisations.

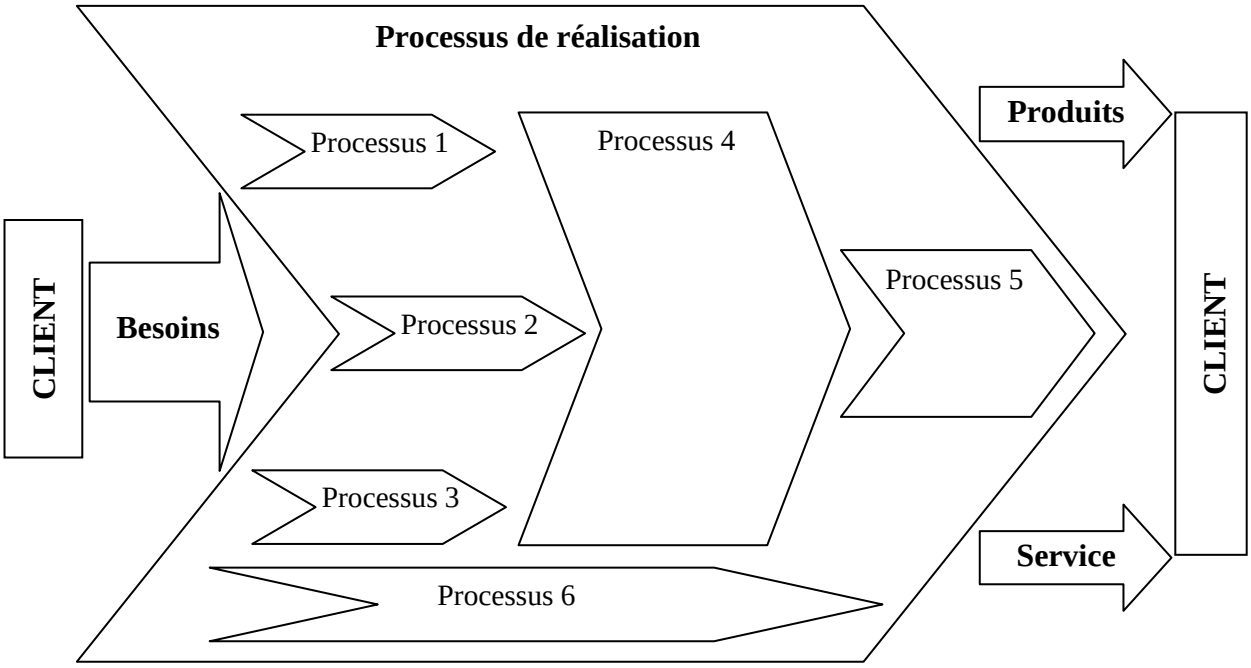


Figure II.4: Composition d'une cartographie de processus de réalisation.

Il existe plusieurs façons d'élaborer cette cartographie. La norme ISO 9001 ne fait aucune référence à une quelconque notation. Il convient d'en utiliser n'importe laquelle. Elle exige cependant que soient décrites, convenablement, la *séquence* et les *interactions* entre les processus.

La notation, actuellement, utilisée par l'entreprise est simple et claire, et est conforme aux exigences de la norme ISO 9000. La représentation, via cette notation, est à la fois verbale et graphique et c'est celle-ci qui servira à la représentation des processus relatifs au métier concerné, dans le cadre de l'étude de l'existant.

2.6. Méthodologie de l'approche processus

L'approche processus est une approche systémique. Cela veut dire, entre autres, qu'il y aura plusieurs niveaux d'analyse. Ce qui est considéré comme *processus* à un niveau d'analyse « N » va devenir *Sous-processus* au niveau d'analyse « N+1 » et vice-versa.

La phase d'identification des processus a un caractère itératif. L'ajout ou la suppression d'un détail (élément) sur la cartographie de niveau « N », risque d'affecter celles de niveau « N+1 » et/ou « N-1 ». Ceci assure la cohérence entre les différents niveaux d'analyses. Toute fois il est possible de voir apparaitre sur la cartographie de niveau « N-1 »

des éléments d'entré ou de sortie (externe) qui n'apparaissent pas nécessairement sur celle de niveau « N » car tout simplement le niveau d'analyse ne permet pas de le voir.

Pour arriver à élaborer une cartographie de processus de n'importe quel niveau, il suffit de savoir comment faire celle du niveau « N-1 ». La méthode est simple et propose de procéder en quatre étapes.

Etape 1: Décrire l'organisme entier comme un macro-processus.

Il convient d'identifier, en premier lieu, le processus global au niveau le plus élevé. La cartographie qui en sortira porte le non de Macro- processus et servira de référence pour les étapes suivantes. Les caractéristiques du processus doivent toutes être mentionnées, notamment, les processus élémentaires (PE) qui correspondent aux activités de l'organisme. Pour plus de clarté, il est conseillé de regrouper les éléments d'entré et sortie par provenance (P) et destination (D). La *figure II.4* est un récapitulatif de ce que doit être en principe la représentation d'un organisme sous forme de macro-processus.

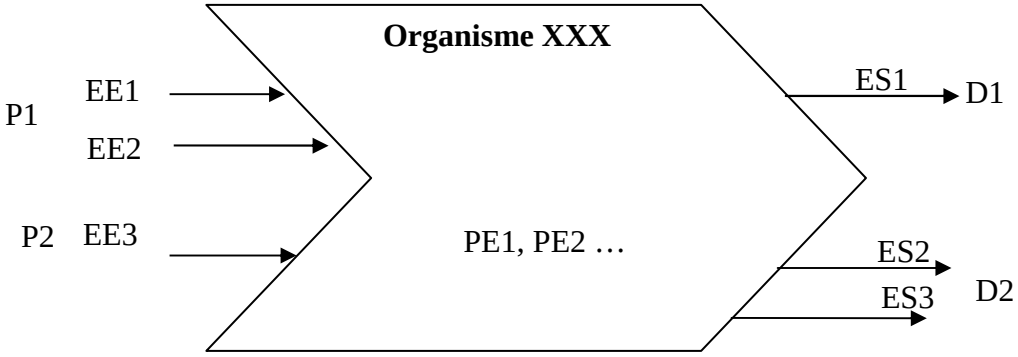


Figure II.5: Représentation d'un organisme sous forme de macro-processus.

Etape 2: Décrire les processus qui prennent en charge les entrées du macro-processus.

Dans cette étape il s'agit d'identifier une partie des processus élémentaires, entre autre, ceux qui prennent en charge les éléments d'entrée du macro-processus. Pour cela, il est conseillé d'aller « sur le terrain » pour suivre, concrètement, auprès des acteurs concernés, qui prend en charge une entrée, quel traitement il effectue, quel est le résultat de ce traitement et où va le résultat de ce traitement.

La *figure II.5* est le résultat de *l'étape1* appliquée sur l'exemple de la *figure II.4*

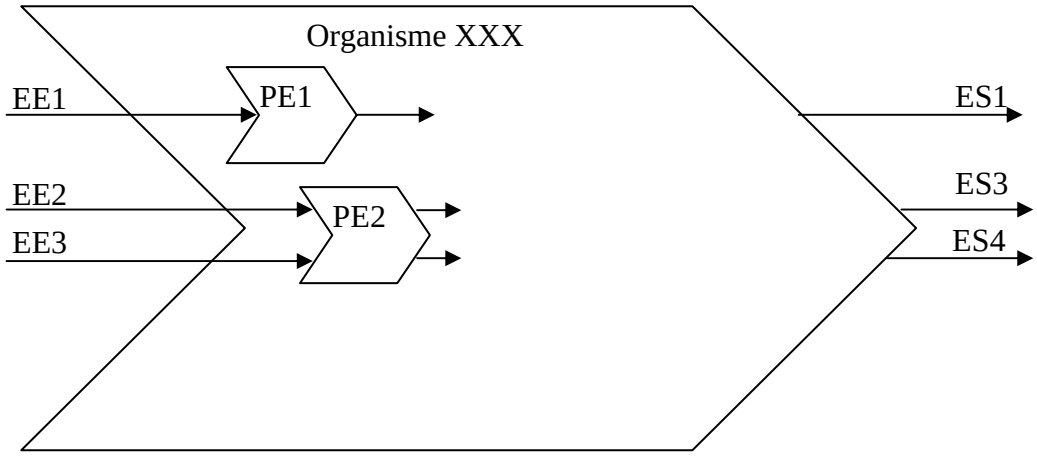


Figure II.6: Exemple de résultat de *l'étape 2*.

Etape 3: Décrire les processus qui génèrent les sorties du macro-processus.

À l'inverse de *l'étape 2*, il s'agit dans celle-ci d'identifier l'autre partie des processus élémentaires, ceux qui génèrent les éléments de sortie du macro-processus. Pour cela, il est conseillé d'aller «sur le terrain» pour remonter, concrètement, auprès des acteurs concernés, qui génère cette sortie, quels traitements il effectue, quelles entrées il prend en charge et quelles sont les origines de ces entrées.

La *figure II.6* est le résultat de *l'étape2* appliquée sur l'exemple de la *figure II.5.*

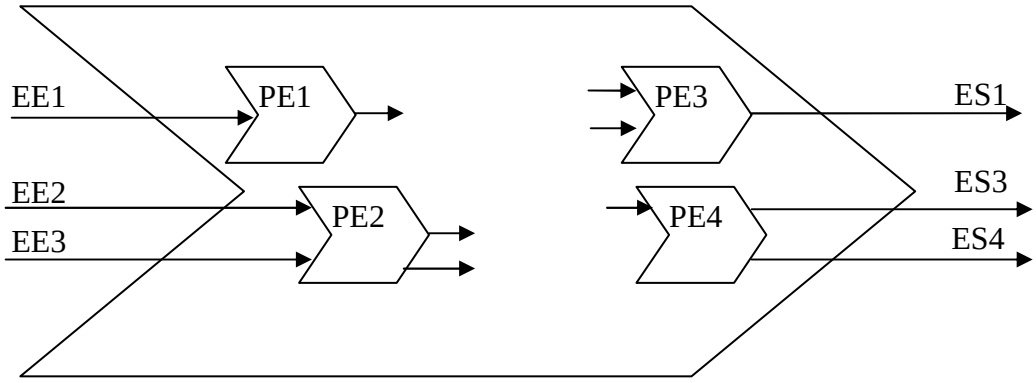


Figure II.7 : Exemple de résultat de *l'étape 3*

Etape 4 : Décrire les processus élémentaires qui manquent dans la chaîne.

Une fois le résultat de *l'étape 2* et celui de *l'étape 3* validé, des éléments de sortie des processus élémentaires issus de *l'étape 2*, ainsi que les éléments d'entrée des processus élémentaires issus de *l'étape 3*, auront était identifiés. Cette étape consiste, soit à prendre chaque élément de sortie interne et à décrire les processus qui le prennent en charge, ou bien, à prendre chaque élément d'entrée interne et à décrire les processus qui le génèrent. Il est fréquent de voir apparaitre des processus particuliers au cours de cette étape. Il s'agit

de processus qui ne sont pas liés directement au processus issus des *étapes 1* et *2*, mais qui ont leurs importances, à l'exemple du processus « PE6 » de la *figure II.5*:

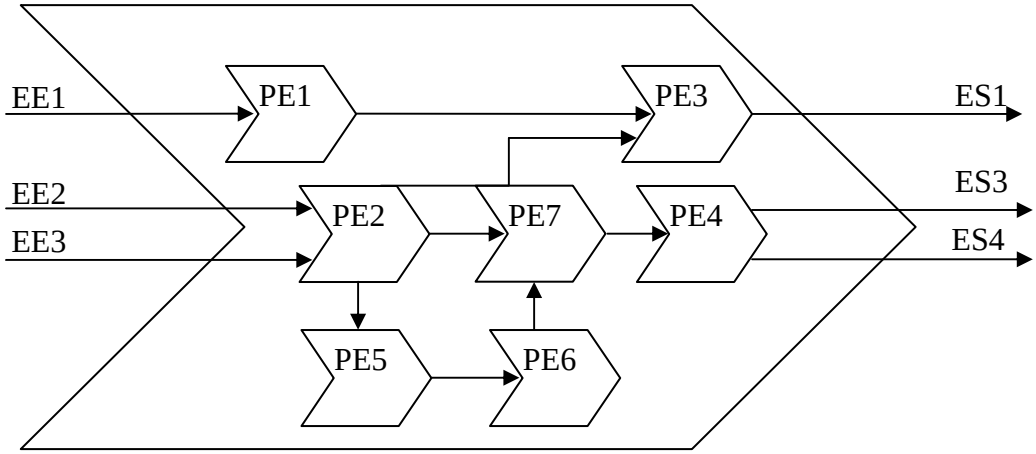


Figure II.8: Exemple de résultat de *l'étape 4*.

Au terme de ces quatre étapes, on aura élaboré la cartographie des processus de réalisation de niveau « N-1 ». Systématiquement, pour réaliser celle de niveau « N-2 » il suffit de prendre chaque processus élémentaire (PE1, PE2, …) à part et de réitérer les *étapes 2,3 et 4* en le considérant comme étant le macro-processus de *l'étape 1*.

Au final, on aura autant de cartographie, qu'il y a de processus élémentaires, qu'il suffira de regrouper dans un même document pour obtenir la cartographie de processus de réalisation de niveau « N-2». Il faut tout de même, prendre le soin de remettre chaque cartographie à la place du processus élémentaires qu'elle est supposée décrire.

Note : Il est recommandé de veiller, constamment, à ce que le résultat, de chaque étape, soit validé par les acteurs concerné avant de passer à une autre. Demander avis aux acteurs, jugés susceptible d'apporter un plus à l'analyse ne ferra qu'enrichir son contenu.

Dans un contexte de management, trouver le bon niveau de détail au quel il convient de s'arrêter est délicat et exige une certaine maîtrise de l'approche. En revanche, dans le contexte d'une simple analyse et modélisation de processus, le bon niveau d'analyse serait celui jugé suffisamment détaillé pour répondre aux besoins de la modélisation.

3. Diagnostic du système informatique existant

Comme nous l'avons vu dans le chapitre I (*Voir le chapitre : Organisme d'accueil*), l'ancien système automatique renferme cinq logiciels à savoir : GAM, ZSD, MSMAN, MS Transit et MS Transport (*Voir figure I.3.Architecture de GLS*).

Dans le cadre de notre travail, il nous a été demandé de diagnostiquer les 3 applications les plus importantes qui englobent le procédé de facturation. Cela, afin de nous permettre de mieux comprendre et cerner son mode de fonctionnement, la politique d'implémentation et de développement de Marine Soft et pouvoir récolter l'ensemble des données manipulées dans le cadre de la facturation. Pour ce faire, nous utiliserons l'approche par processus.

Le schéma suivant délimite le champ d'étude et de diagnostic

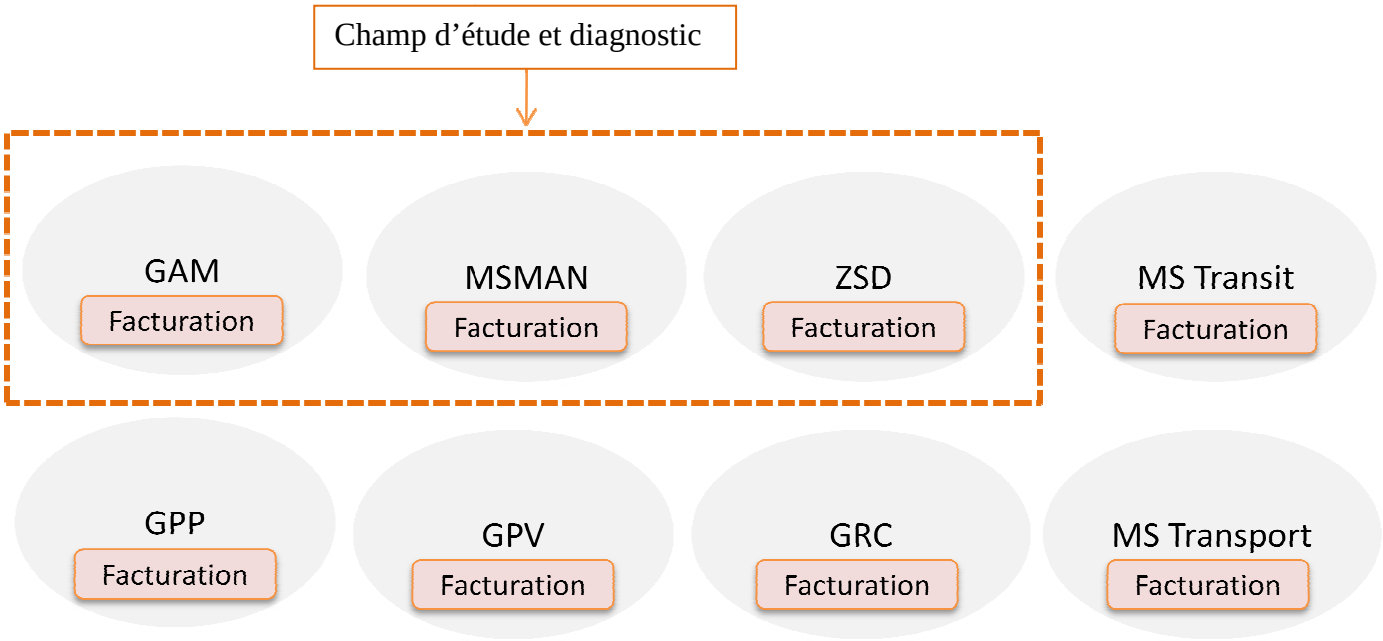


Figure II.9 : Champ d'étude et diagnostic.

Remarque

L'utilisation de la loi de Pareto[2] permet de déterminer rapidement quelles sont les priorités d'actions. Si on considère que 20% des causes présentent 80% des occurrences, agir sur ces 20% aide à solutionner un problème avec un maximum d'efficacité.

Cette méthode peut s'appliquer pour justifier le choix de notre champ d'étude, puisque 80% des fonctionnalités existent dans 20% des applications de Marine Soft :

[2] Livre : Bien vivre le principe 80/20 par Richard Koch, 2011.

- ✓ ***GAM*** *: c'est le logiciel le plus complexe et le plus complet, il couvre toutes les notions de facturation.*
- ✓ ***ZSD*** *: Marine Soft veut investir dans ce progiciel pour des raisons commerciales, il a la même logique de conception avec GAM.*
- ✓ ***MSMAN*** *: La logique de la conception du module facturation est presque la même pour tous les progiciels de Marine Soft à l'exception MSMAN d'après les ingénieurs, de ce fait on l'a choisi pour l'analyser.*

Pour arriver à nos fins, nous avons suivi la procédure des interviews, qui nous a permis de contacter directement les concernés.

La démarche adoptée se résume comme suit :

- ➢ Fixer les objectifs que nous devons atteindre.
- ➢ Faire des interviews.
- ➢ Analyser les informations recueillies et proposer des solutions.

Cette étape comprend :

- Définition des produits de Marine Soft.
- L'architecture de chaque produit et la répartition du code.
- L'analyse de chaque produit par la méthode **« approche processus ».**
- La description des tables de la base de données de chaque produit **« module facturation ».**
- La description des modules de chaque produit.

Remarque

Dans l'étape « Analyse de chaque produit par la méthode approche processus », le niveau N-1 est ignoré dans toute notre étude, car celui-ci présente des modules qui ne nous intéressent pas. Nous allons donc ainsi directement du niveau N au niveau N-2.

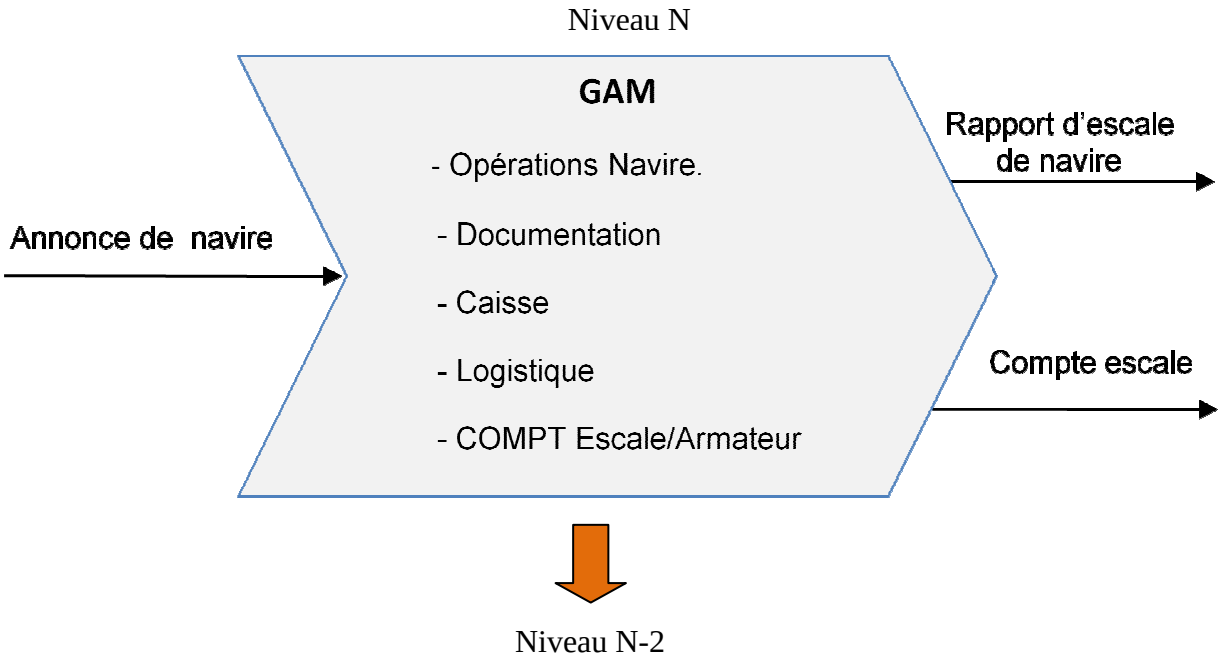
3.1. Etude du logiciel GAM (Global Agency Management)

3.1.1. Définition du logiciel

GAM est un progiciel qui s'adresse aux agents consignataires de navires. Il permet d'automatiser du début de la chaîne de traitements jusqu'à la fin, les opérations de gestion quotidienne à savoir :

- Facturation (établissement des factures BL et Surestaries, gestion des factures Avoirs et Complémentaires, suivi des règlements des factures des clients conventionnés, établissement des états de caisses, transfert des factures vers la comptabilité, …etc.).
- Le suivi des Opérations Navire.
- La documentation (saisi du manifeste, taxation des B/L,…).
- La déclaration douanière.
- Le suivi des situations BL.
- La gestion des conteneurs et des comptes armateur.

3.1.2. Présentation de GAM sous forme de processus



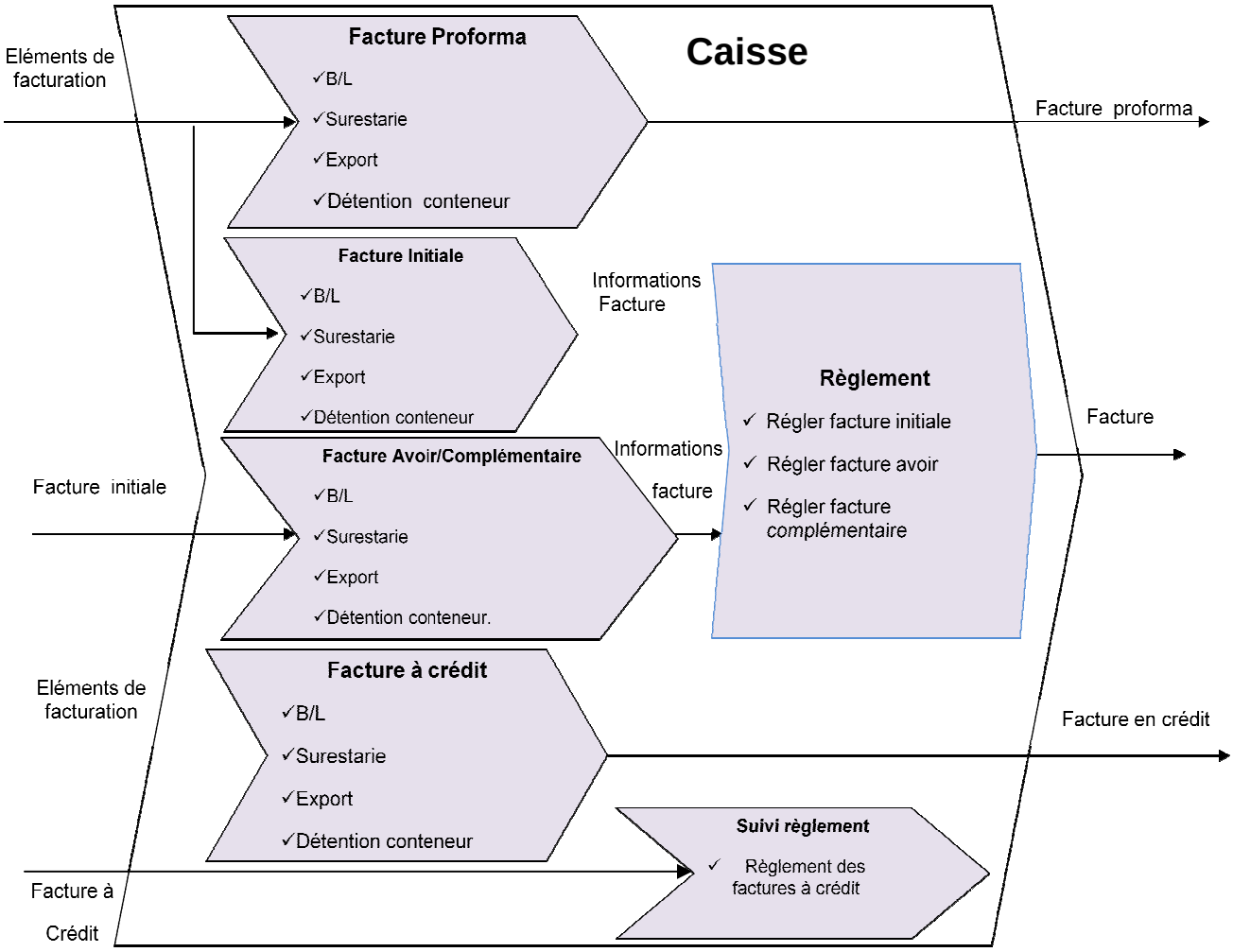


Figure II.10. Présentation de GAM sous forme processus.

3.1.3. Architecture et la répartition du code du logiciel GAM

Le logiciel GAM s'appuie sur l'architecture client/serveur 2tiers. Son code est réparti comme suit :

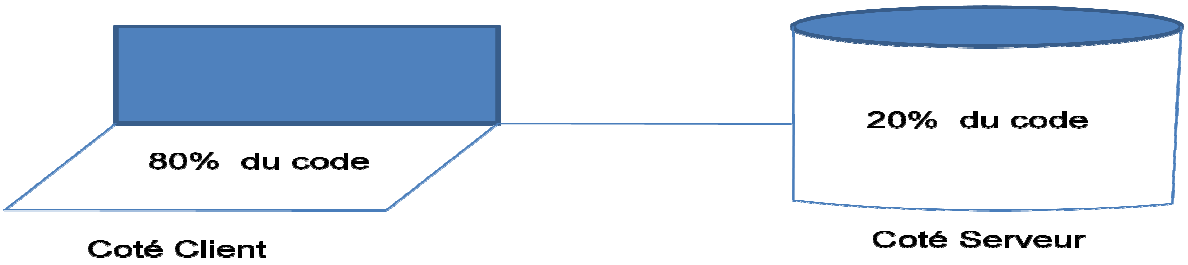


Figure II.11: L'architecture et la répartition du code du logiciel GAM.

3.1.4. Description des modules de GAM

Modules	Description
	Permet d'accéder aux fichiers et le paramétrage de tout le progiciel.
	Permet d'accéder aux fichiers de bases (type d'emballage, type navire, … etc.).
	Assure la gestion opérationnelle (dossiers, situations portuaires, … etc.).
	Permet la saisie des B/L, l'édition des avis d'arrivées, l'édition des fiches de recette pour les opérations d'importation, … etc.
	Permet la saisie des booking, la taxation des B/L, l'édition des pro format d'exportation, l'édition des fiches de recette pour les opérations d'exportation.
	Permet l'établissement du manifeste douanier, l'établissement du manifeste rectificatif, … etc.
	Permet l'établissement du bon à délivrer, le suivi des situations des B/L, … etc.
	Permet l'établissement des factures avoirs, B/L, complémentaires et surestaries, l'édition de l'état de caisse, …
	Assure le suivi des conteneurs, la gestion des mouvements des conteneurs, la gestion des proformas de surestaries, suivi des cautions, … etc.
	Permet l'établissement des comptes des dossiers, l'édition des notes de débit douane, des comptes dossiers complémentaires, …etc.
	Permet l'établissement des comptes courants dossiers (compte armateur), la gestion des comptes dossiers transférés à l'armateur,… etc.
	Permet l'échange informatisé des données entre le progiciel GAM® 2.7.6 et les autres logiciels armateurs, … etc.

Tableau II.1. Les modules de GAM

3.1.5. Les tables de la base de données

Dans cette partie notre étude sera restreinte aux tables de la facture B/L et Surestarie :

3.1.5.1. Les tables de la facture B/L

Une facture B/L est une facture à l'import, contenant tous les frais des prestations et des débours (les frais à payer par l'agence consignataire à la place du client) engendrés par la marchandise que le réceptionnaire doit payer à l'agence consignataire.

Le schéma suivant représente la base de données de la facture B/L :

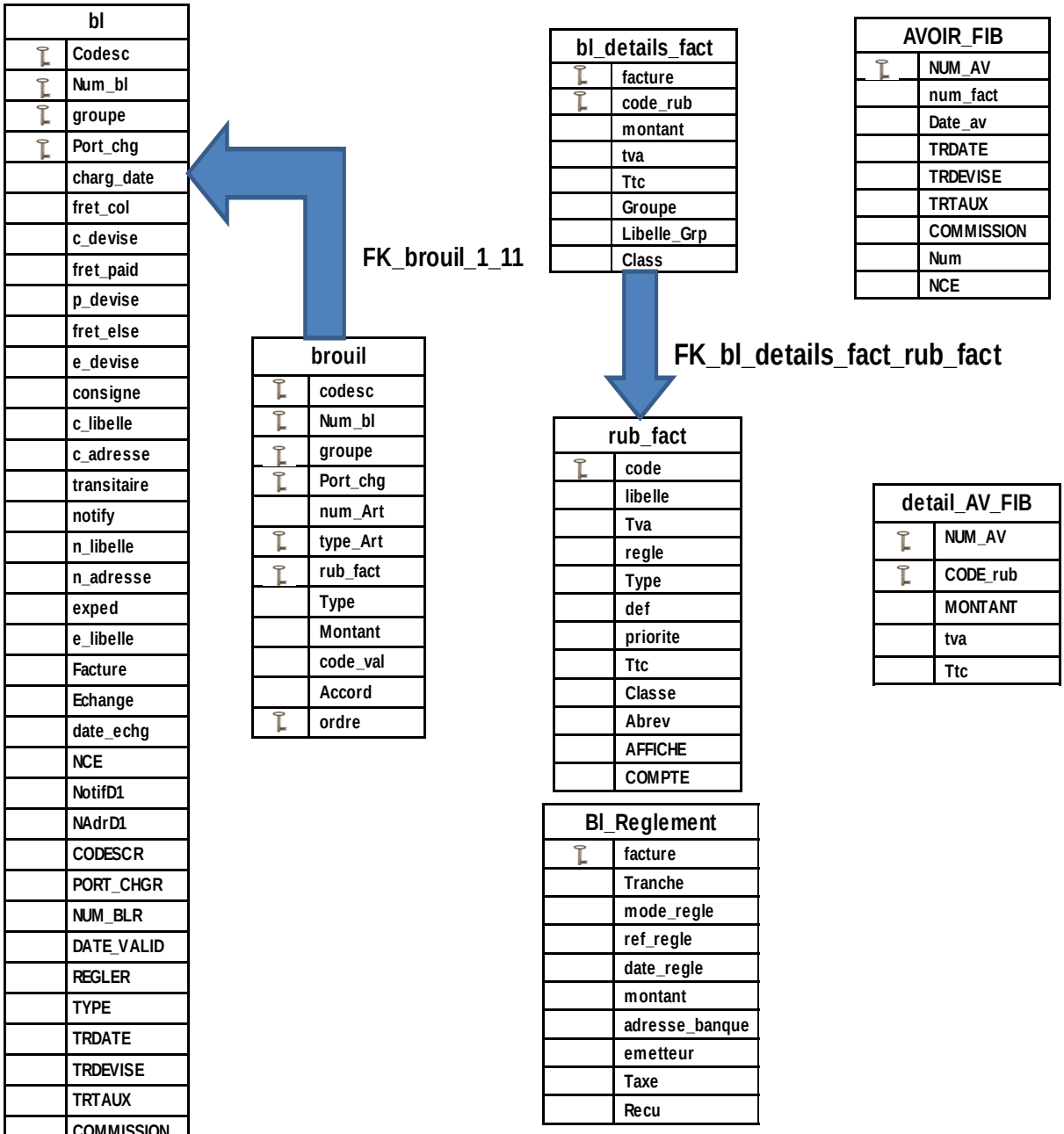


Figure II.12 : Les tables de la facture B/L

3.1.5.2. Les tables de la facture Surestarie

Une facture surestarie est une facture à l'import, contenant les frais d'allocation des conteneurs à partir du débarquement que le réceptionnaire doit payer à l'agence consignataire.

Le schéma suivant représente la base de données de la facture Surestarie :

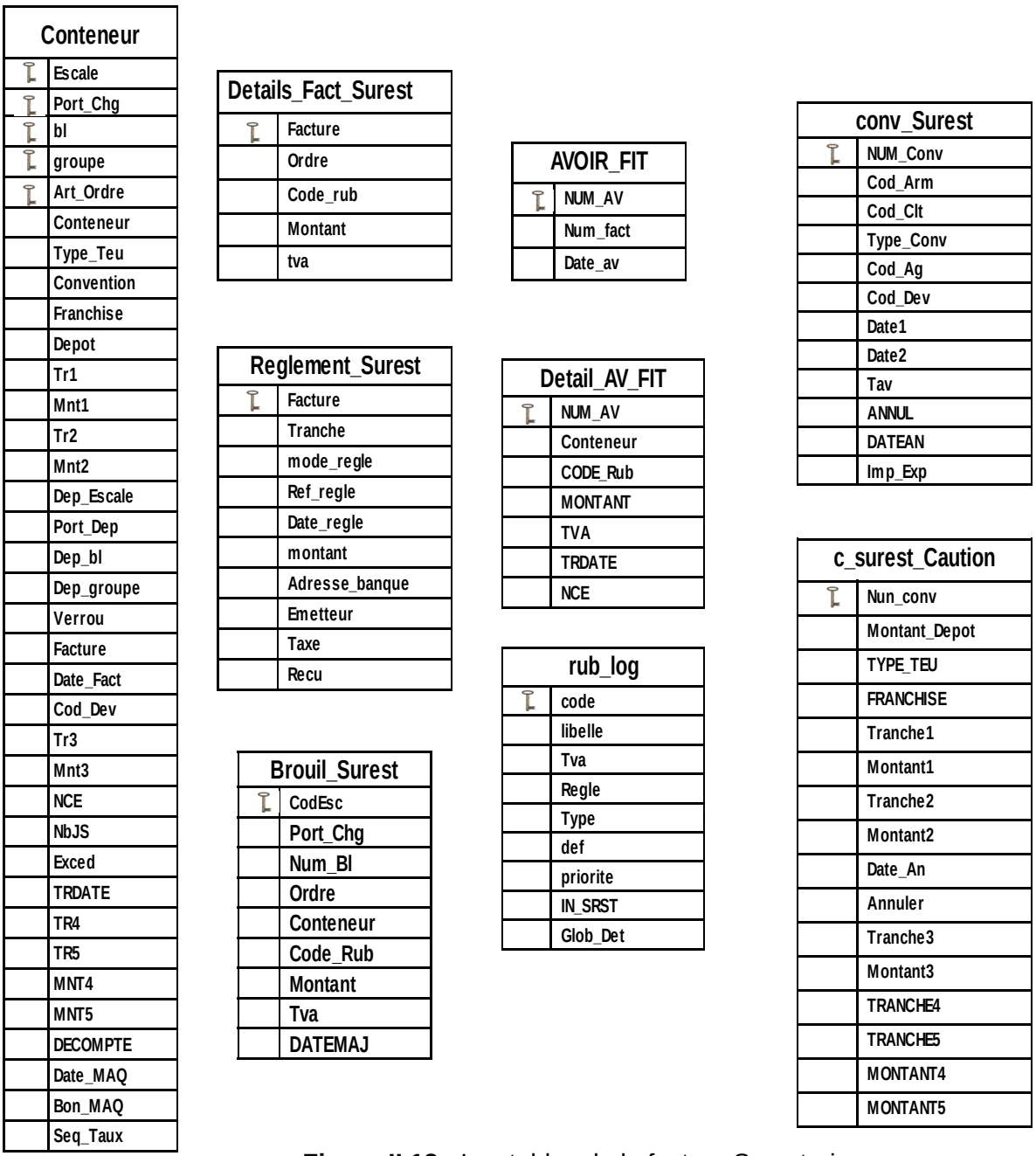


Figure II.13 : Les tables de la facture Surestarie.

3.2. Etude du logiciel MS MANUTENTON

3.2.1. Définition du logiciel

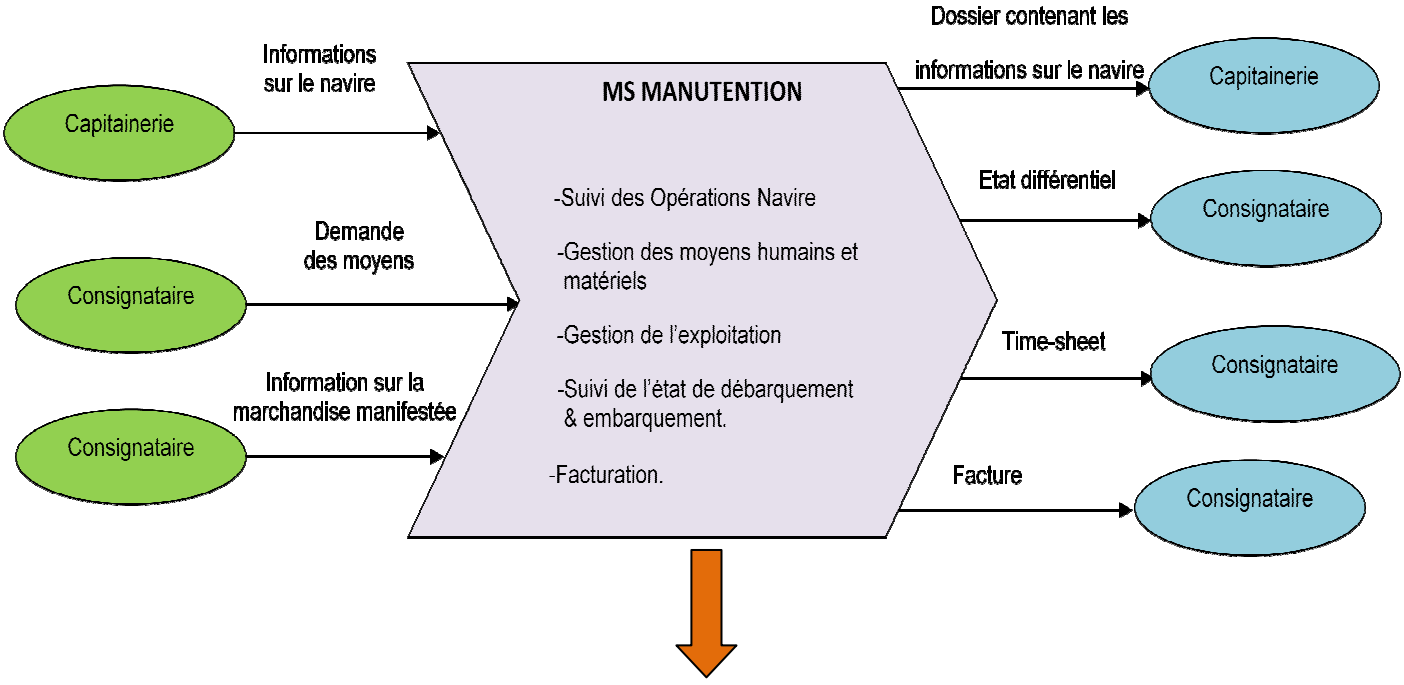
MSGMAN est un progiciel qui s'adresse aux manutentionnaires. Il permet d'automatiser du début de la chaîne de traitements jusqu'à la fin, les opérations de gestion quotidienne à savoir :

- Suivi des Opérations Navire (création & suivi de l'escale, édition des situations portuaires..).
- Gestion des moyens humains et matériels (commandes, programmations et affectations ainsi que le pointage des ouvriers).
- Gestion de l'exploitation (pré - régime, Time-sheet….).

  Suivi de l'état de débarquement & embarquement (Manifeste cargo, état du débarquement, état différentiel).
- Facturations (Etablissement des factures : débarquement, embarquement, pointeur, Extra-frais, transfert et location-grues, Gestion des conventions, gestion du transfert des factures vers la comptabilité).
- Statistiques.

3.2.2. Présentation de MSMAN sous forme de processus

Niveau N



Niveau N-2

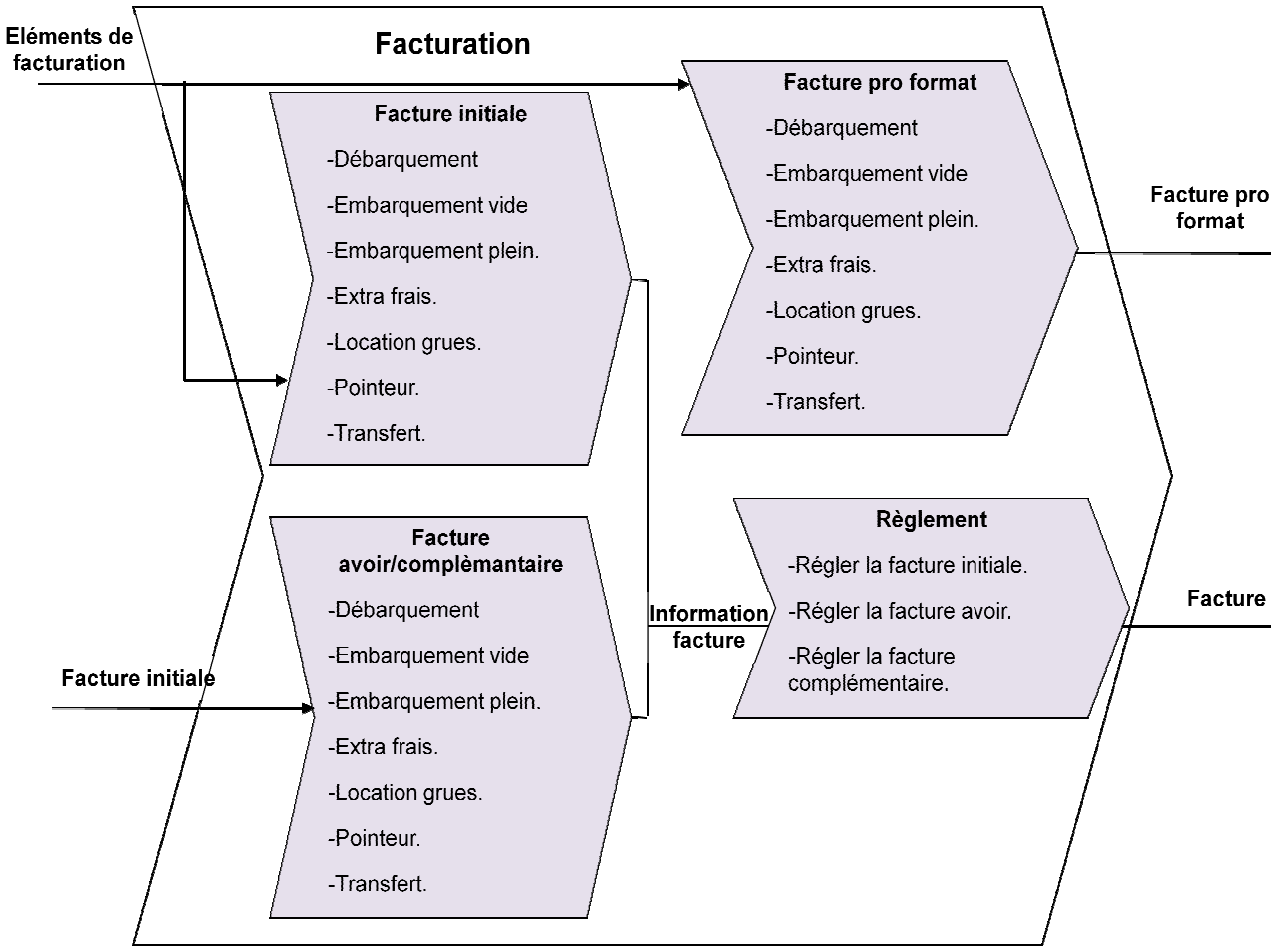


Figure II.13: Présentation de MSMAN sous forme de processus.

3.2.3. Architecture et la répartition du code du logiciel MSMAN

Le logiciel MSMAN s'appuie sur l'architecture client/serveur 2tiers. Son code est réparti comme suit :

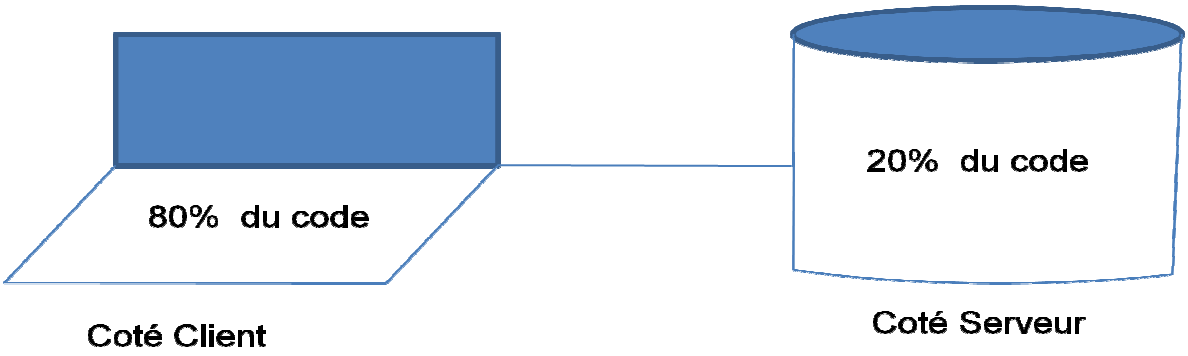


Figure II.14: L'architecture et la répartition du code du logiciel MSMAN.

3.2.4. Description des modules

Modules	Descriptions
	Permet d'accéder aux fichiers permettant le paramétrage de tout le progiciel.
	Permet d'accéder aux fichiers de bases (navires, agents, type navire, … etc.).
	Permet de pointer les ouvriers.
	Permet l'établissement des factures : Extra-frais, transfert, pointeurs, location-grues, débarquement et embarquement. Les factures avoirs complémentaires, l'édition de l'état de caisse, journal des factures… etc.
	Production et exploitation des états statiques permettant le bon suivi des opérations de traitement des navires.
	Permet l'échange informatisé des données entre le progiciel MSGMAN et les autres logiciels GESPort, … etc.

Tableau II.2. Les modules de MSMAN

3.2.5. Les tables de la base de données

Le schéma suivant présente les tables de données du module facturation de MSMAN :

Figure II.15 : Les tables du module facturation de MSMAN.

Grâce au champ « **TYPEF** »qui désigne le type de la facture, on peut identifier sept types de factures : Débarquements, Embarquement plein, Extra frais, Transferts, Location grues, Pointeurs, Embarq vide.

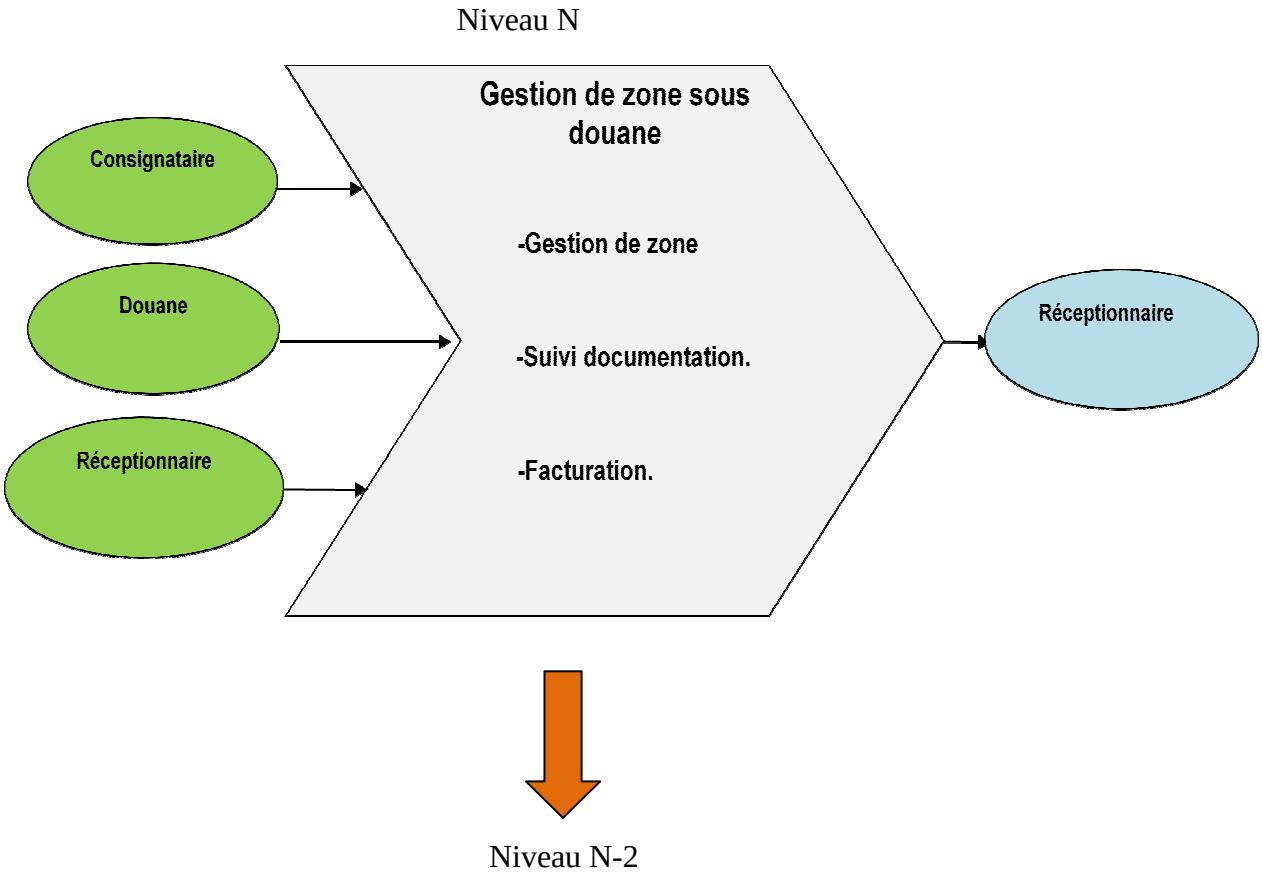
3.3. Etude du logiciel ZSD

3.3.1. Définition du logiciel

ZSD est un progiciel qui s'adresse aux entrepôts sous douane. Il permet d'automatiser la gestion de l'entrepôt et le stockage des marchandises à savoir :

- Suivi de la documentation (saisi des ordres de transfert, DSTR),
- Gestion de la zone (Saisie des entrées et sortie des articles, gestion du parc…)
- Facturation (Établissement des factures armateur, établissement des factures client, gestion des factures, …).

3.3.2. Présentation de ZSD sous forme de processus



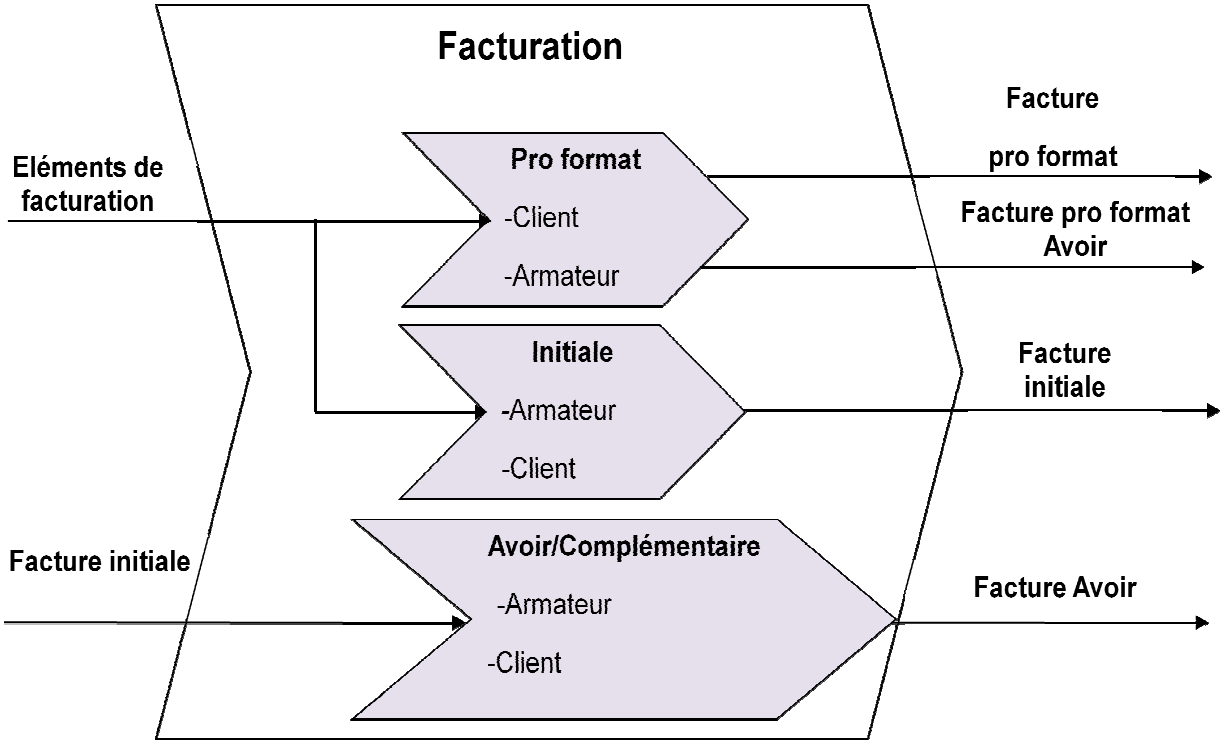


Figure II.16: Présentation de ZSD sous forme de processus.

3.3.3. Type d'architecture et la répartition du code du logiciel ZSD:

L'architecture du logiciel ZSD s'appuie sur le mode client/serveur (2tiers). Son code est réparti comme nous allons l'illustrer dans le schéma qui suit :

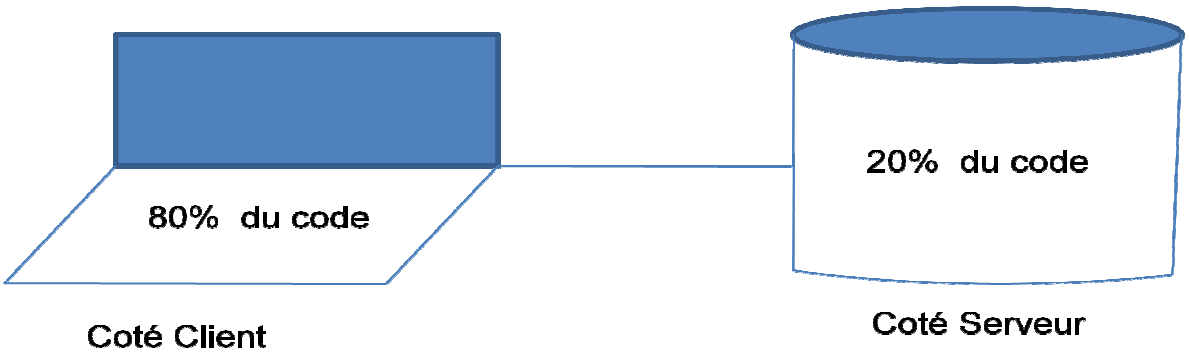


Figure II.17: L'architecture et la répartition du code dans le logiciel ZSD.

3.3.4. Description des modules

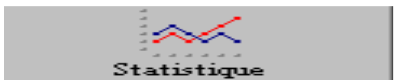
Modules	Description
	Permet d'accéder aux fichiers d'Administration (Conventions, Rubriques Facturation).
	Permet d'accéder aux fichiers de référence (Armateur, Agent, Navire, etc.).
	Permet la saisie des manifestes D1 des articles à mettre en Zone Sous Douane.
	Permet la gestion des entrées /sorties des conteneurs dans la Zone Sous Douane.
	Permet la facturation automatique des conteneurs livrés de la Zone Sous Douane.
	Offre des fonctions de maintenance et de sauvegarde des données du progiciel.
	Permet de visualiser la situation de la Zone avec une vue en trois dimension (réelle).
	Offre des fonctions de transfert et de consolidation des données du progiciel.
	Offre au gestionnaire de la Zone, des statistiques concernant le Parc.
	Offre à l'utilisateur de la Zone, l'aide contextuelle concernant le Progiciel.

Tableau II.3. Les modules de MSMAN

3.3.5. Les tables de la base de données

Cette base de données nous permet de concevoir deux types de facture : la facture Client et la facture Armateur qu'on va définir ci-dessous :

3.3.5.1. Facture Client

Une facture Client est une facture qui contient tous les frais des prestations et des débours (les frais à payer par la zone sous douane à la place du réceptionnaire) engendrés par les conteneurs que le réceptionnaire doit payer à la zone sous douane.

3.3.5.2. Facture Armateur

Une facture Armateur est une facture qui contient des prestations et des débours (les frais à payer par la zone sous douane à la place de l'armateur) que l'armateur doit payer à la zone sous douane.

Le schéma suivant représente la base de données du module facturation de ZSD :

Figure II.18 : Les tables du module facturation de ZSD.

4. Bilan du diagnostic (anomalies et causes)

Suite au diagnostic effectué, sur les trois principales applications, à savoir : GAM, ZSD, MSMAN, nous avons constaté que ces dernières imposent un ensemble de contraintes dues essentiellement à deux causes à savoir : l'architecture 2tiers et la duplication du composant applicatif de facturation :

1. Architecture 2 tiers :

- Le système informatique de Marine Soft est essentiellement basé sur une architecture de type client-serveur (2-tiers[1]). Nous avons constaté que ce type d'architecture était très contraignant pour des opérations de maintenance logicielle et de déploiement, car la charge du poste client supporte la grande majorité des traitements applicatifs. En effet à chaque modification (mise à jour) ou un ajout d'une nouvelle fonctionnalité dans le module facturation, il fallait redéployer la nouvelle version logicielle au niveau de chaque poste client.
- Les applications sont supportées par un seul système d'exploitation WINDOWS, et par un seul type d'équipement (PC).

2. Duplication du composant applicatif de facturation dans l'ensemble des progiciels :

Comme nous l'avons présenté dans le chapitre I: Organisme d'accueil (*Voir figure I.1. Architecture de **GLS***), nous avons constaté que le module facturation est dupliqué dans tous les progiciels: (GAM, ZSD, MSMAN, MS Transit, MS Transport).Cette situation a engendré plusieurs problèmes :

1. Dépendance du système de facturation de la logique métier

Nous avons constaté, que le module facturation qui existe dans toutes les applications dépend du progiciel qui l'intègre. Autrement dit, ne nous pouvons pas dissocier la facturation du métier (chaque métier possède un système de facturation propre à lui). Cette dépendance rend la facturation non standard.

[1] *Voir annexe I*

Exemple

Le système de facturation dans l'application ZSD est lié à sa logique métier. On le constate dans la partie données et dans la partie traitements.

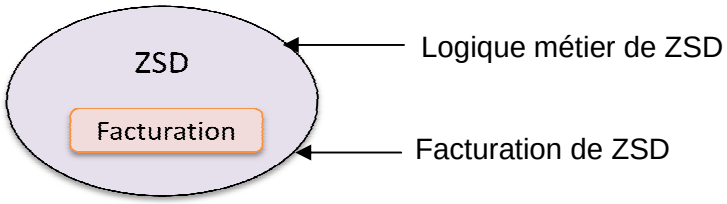


Figure II.19 : Dépendance du système de facturation de ZSD

2. Maintenabilité lourde des applications dans le système de facturation

Lors de la maintenance du système de facturation, il faut maintenir tous les systèmes de facturation des applications ainsi que tous les postes clients.

Exemple

Si le procédé de facturation change, on est obligé de le mettre à jour dans toutes les applications et dans chaque poste client.

3. Redondance des champs

Il y a plusieurs champs qui signifient la même chose mais qui sont appelés différemment d'une application à une autre.

Exemple

- numFac dans le progiciel ZSD.
- NFACTURE dans le progiciel MSMAN.
- Facture dans le progiciel GAM.

Ce champ signifie le numéro de la facture dans les trois applications.

3. Redondance des tables

Chaque application a sa propre conception de base de données. D'où la redondance des tables.

5. Présentation de la solution informatique

Comme nous l'avons cité dans le chapitre I (Organisme d'accueil), le projet actuel de Marine Soft est le développement de **l'ERP GLS**, qui va pallier au problème de l'ensemble du système informatique.

5.1. Présentation de la solution ERP GLS

L'ERP GLS est destiné à la logistique et le transport de la marchandise, et contient plusieurs modules que nous illustrons dans le schéma suivant :

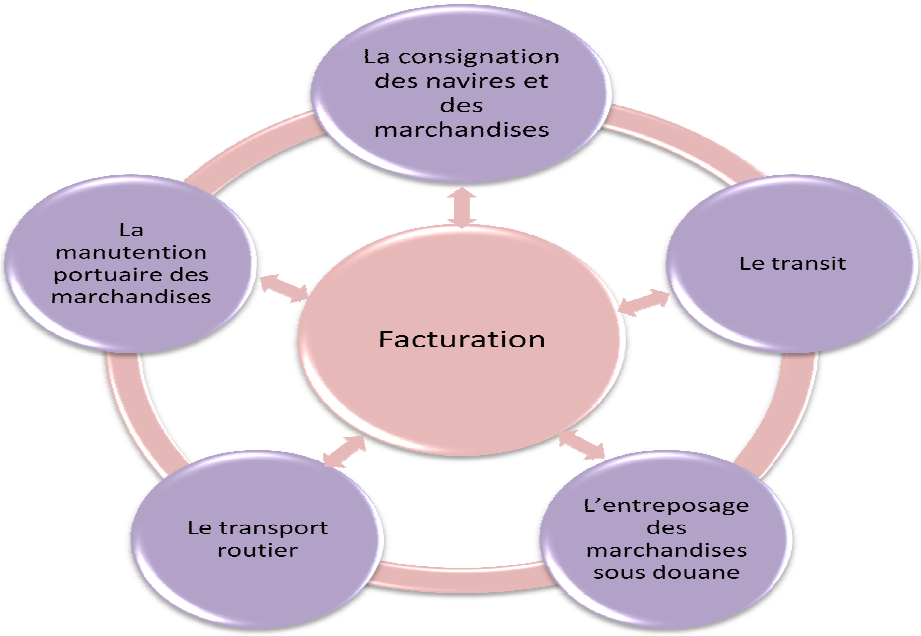


Figure II.20: La solution GLS.

La fiche d'identité de l'ERP GLS est reprise dans ce qui suit :

La désignation	GLS (Global Logistic Software).
Type	ERP.
Composant métier	- Le transit. - La consignation des navires et des marchandises. - La manutention portuaire des marchandises. - Le transport routier. - L'entreposage des sous douane.
Noyau	**La facturation**
Le langage de programmation	JAVA
Architecture	Client/serveur 3tiers.
Utilisateur	Agences consignataires, transitaires, ports, ports secs, entrepôts.

Tableau II.4. La fiche d'identité de **GLS.**

5.2. Les objectifs de l'ERP

Parmi les objectifs principaux de l'ERP GLS nous citons :

1. **Basculer vers une architecture 3 tiers de type J2EE**

Pour rendre plus opérationnelle et plus performante l'architecture logicielle de l'ERP, il a été décidé de basculer vers une architecture 3tiers de type J2EE[3] (*Voir figure II.21*). Ainsi, les applications seront supportées par tous les systèmes d'exploitation et par tout type d'équipements: PC, PDA, mobile...Etc.

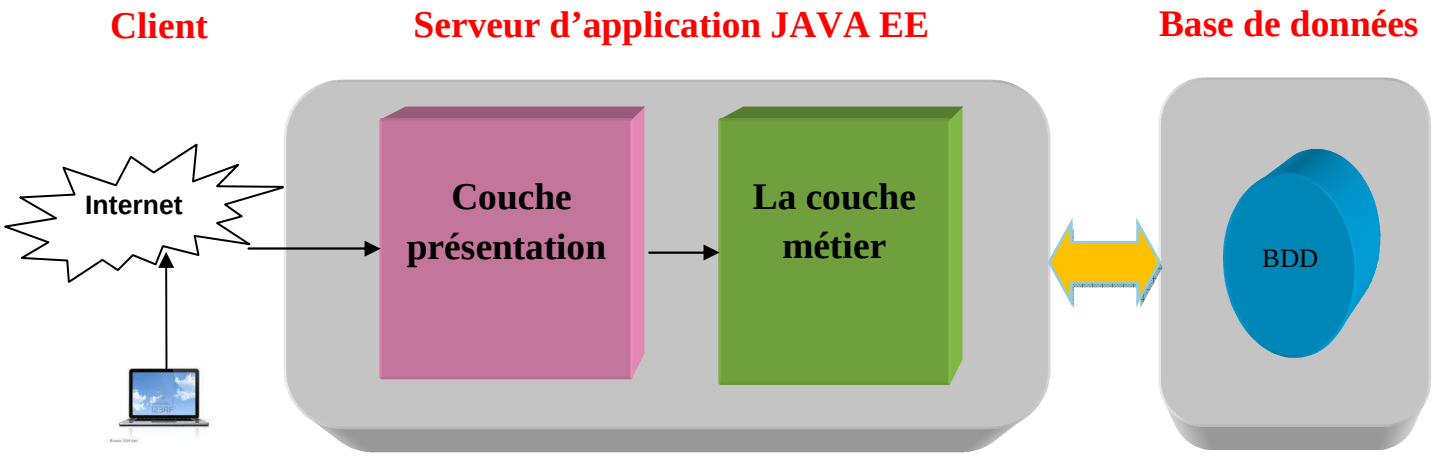


Figure II.21. Architecture 3tiers de type Java EE.

2. **Elimination de la dépendance**

Dissocier la logique métier de la logique de facturation, pour aboutir, à une conception d'un système de facturation unifié pour toutes les applications métiers de l'ERP GLS.

3. **Elimination de la redondance des données dans la base de données**

- Mettre en place une base de données centralisée.
- Eliminer la redondance des tables qui apparaissait dans les anciennes applications de Marine Soft.
- Uniformiser les appellations des données pour une **application standard**.

D'où une meilleure cohérence et une plus grande optimisation dans la gestion des données.

[3] *Voir annexe I et annexe II.*

4. Maintenance aisée

Réunir **les modules de facturation dupliqués** dans une seule application pour le système de **facturation unique**. En cas de modification (mise à jour). On ne sera pas obligé de maintenir tout les modules de facturation existants dans les applications, il suffit seulement de maintenir le nouveau système de facturation.

L'ERP présente de plus, d'autres objectifs qui sont :

- Une architecture fonctionnelle **modulaire**, permettant **l'exportabilité** vers d'autres systèmes.
- Une séparation des fonctions techniques (données, métier, interfaces utilisateur et administration).
- L'organisation modulaire permettra d'une part, l'évolutivité des systèmes en fonction des progrès technologiques et d'autre part l'accès aux grandes fonctions du système.

L'ERP GLS intégrera les grandes fonctionnalités déjà existantes dans les différents produits Marine Soft, en particulier :

- **Automatisation des procédures documentaires maritimo-portuaires et des échanges d'information entre acteurs.**
- **Facturation entièrement paramétrable.**
- **Etats de sortie paramétrables.**
- **Passerelles vers des logiciels de compatibilité** : une interface paramétrable pour faire la correspondance des rubriques de facturation avec les codes comptable.
- **Connectivité** : se connecter simultanément à plusieurs sites, plusieurs serveurs et plusieurs bases de données tout en choisissant le type d'authentification et le protocole de connexion.
- **Remontée des alertes** : baser sur des protocoles de messagerie (SMTP, SMS etc.…).

Rappel

On rappelle que dans le cadre du projet GLS, Marine Soft nous a confié la conception et la réalisation du noyau **« Facturation »**, ainsi que la mise en place d'un système de communication avec les autres applications de GLS.

La gestion de la facturation représente une partie importante du noyau de l'ERP GLS, c'est un module commun à toutes les applications métier, nous citons :

1. La consignation des navires et des marchandises.
2. La manutention portuaire des marchandises.
3. L'entreposage des marchandises sous douanes (entrepôts publics, ports secs et ports).
4. Le transport routier.
5. Le transit.

Les prestations fournies par les applications métier exerçant les activités du transport citées ci-dessus, sont automatiquement facturées aux clients, d'où l'intérêt du système d'information en question.

Conclusion

L'analyse approfondie des progiciels de Marine Soft nous a permis de mettre en évidence certaines imperfections aussi bien au niveau des logiciels applicatifs que dans celui de l'architecture logicielles mise en œuvre. Ainsi nous avons proposé une solution à adopter pour résoudre notre problématique en présentant les objectifs de l'ERP GLS. Nous passerons dans ce qui suit, à la conception de cette solution pour enfin la réaliser.

Partie II

Analyse & conception

Chapitre I : Analyse & conception

- ¨Présentation d'UML
- Analyse
- Modélisation de l'aspect dynamique de la solution proposée
- Modélisation de l'aspect statique de la solution proposée

Pour d'une meilleure organisation et une bonne maîtrise du travail, tout processus de développement d'applications ou systèmes informatiques doit suivre une méthode ou démarche bien définie.

Dans ce chapitre, en premier lieu, nous allons analyser le processus pour mettre en évidence les différents acteurs intervenants dans le système cible ainsi que leurs besoins. En suite, la phase modélisation de l'aspect dynamique s'appuie sur les résultats de la phase analyse, donnera la modélisation des objectifs à atteindre. Pour ce faire, nous avons opté pour une démarche de conception orientée objet, et enfin, la phase de modélisation de l'aspect statique consiste à la modélisation des données que le système futur utiliserait, en se basant sur le langage de modélisation UML (*voir la figure I.3*).

La figure suivante montre la représentation graphique de la démarche de modélisation que nous avons choisie pour concevoir notre application.

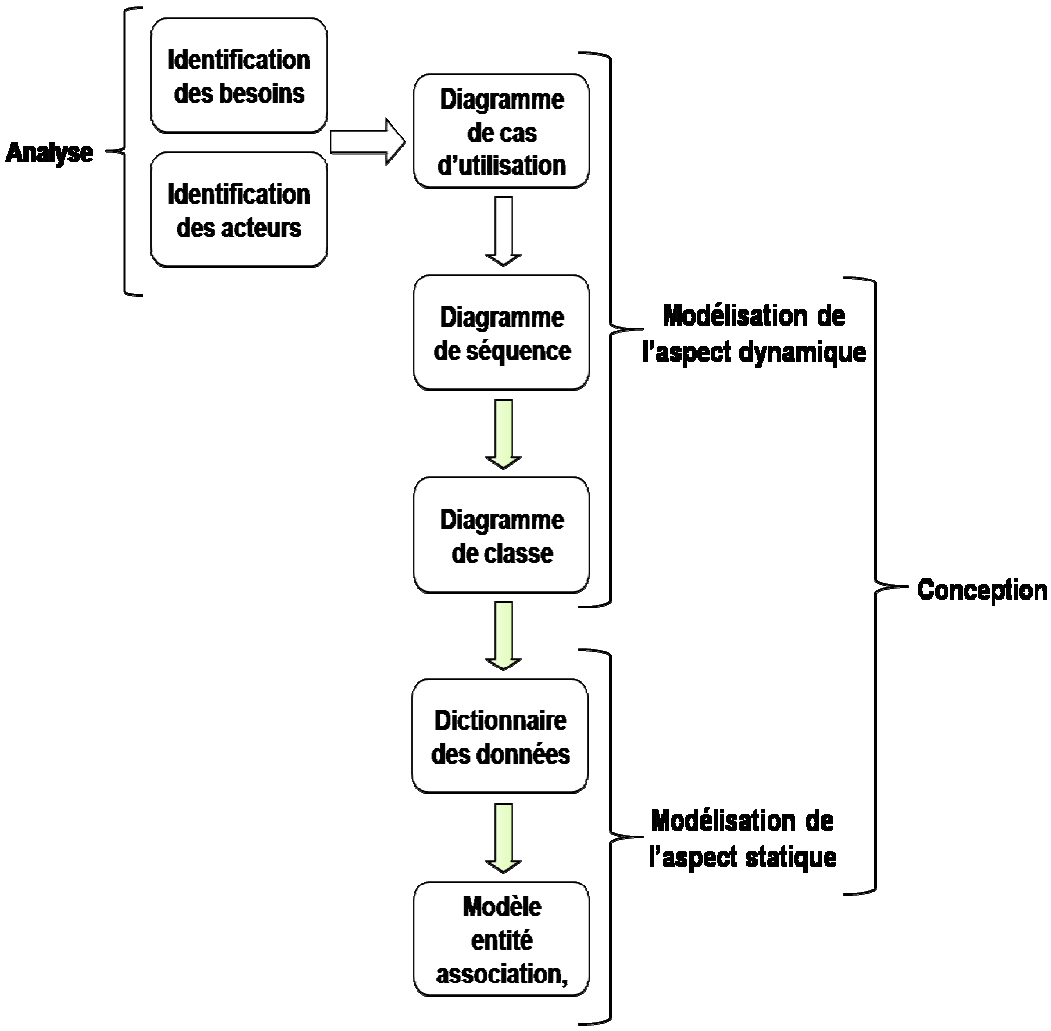


Figure I.1: Représentation graphique de la démarche suivie pour la modélisation de la solution.

1. Présentation de l'UML

UML (Unified modiling language « langage de modélisation objet unifié ») est un langage ou formalisme de modélisation orienté objet qui représente un moyen de spécifier et de représenter les composants d'un système informatique. UML est un standard car à partir de 1997, il est devenu une norme de l'Object Management Group (OMG).

Parmi les objectifs d'UML : être indépendant des langages de programmation et être adapté à toutes les phases du développement. Il est conçu pour la visualisation, la spécification et la construction des systèmes logiciels.

2. Analyse

Dans le but de spécifier d'une manière détaillée et précise les interactions significatives (interactions entre le système et les acteurs), du point de vue de l'application, nous devons d'abord recenser les objectifs et besoins de notre application. Nous identifierons par la suite les acteurs de notre application, ainsi les règles de gestion.

2.1. Identification des besoins

Selon le profil de l'utilisateur connecté à l'ERP, l'IHM de notre application doit être entièrement paramétrable pour s'adapter à chaque activité des applications métiers.

Cette application doit également prendre en charge les fonctions suivantes :

1. Facture pro format, initiale, avoir et complémentaire.
2. Un outil de recherche multi critères des factures et des règlements.
3. Le suivi des règlements des factures à crédit.
4. Passerelle vers le logiciel de comptabilité.
5. Une couche de communication avec les autres modules de l'ERP.
6. L'IHM de l'application doit être entièrement paramétrable.
7. Multilingue.

2.2. Identification des acteurs

Un acteur représente un rôle que peut jouer l'utilisateur dans le système. Un acteur est schématiquement représenté par un personnage stylisé sur les diagrammes UML. Les acteurs de notre système sont :

- **Utilisateur** : personne possédant les droits d'accès à toutes les fonctionnalités du système.
- **Administrateur :** personne possédant les droits de gérer les comptes utilisateurs.

2.3. Identification des règles de gestion

La règle de gestion est la traduction conceptuelle des objectifs choisis et des contraintes acceptées par l'entreprise. Elle est plus particulièrement liée aux traitements (règle d'action) ou aux données (règle de calcul).

Quelques règles de gestion :

- Les montants de la facture initiale doivent être les mêmes avec ceux de la facture pro format.
- La facture initiale ne doit jamais être modifiée.
- La facture pro format n'a pas une valeur comptable.
- L'enregistrement de la facture se fait après le règlement de la facture.
- La modification du montant d'une rubrique de la facture initiale (Avoir ou Complémentaire) se fait sur le dernier montant de la rubrique qui sera calculé comme suit : montant de la rubrique= (montant de la rubrique de la facture initiale + ∑ montants de la rubrique dans les factures complémentaires + ∑ montants de la rubrique dans les factures Avoirs).
- Les montants de la facture initiale, la pro format et complémentaire doivent être supérieur à zéro.
- Le montant d'une facture avoir est toujours négatif.
- L'existence des factures avoirs et complémentaires est relative à l'existence de la facture initiale.
- La facturation à crédit se fait uniquement pour les clients conventionnés.
- Dans le module état de caisse, la date fin est supérieure à la date début.
- Dans la facture complémentaire, on peut rajouter une ou plusieurs rubriques.
- Pour les clients non conventionnés, la totalité de la facture doit être réglée avant l'enregistrement dans la base de données.

3. Modélisation de l'aspect dynamique de la solution proposée

Le processus de modélisation de l'aspect dynamique décrit les fonctionnalités et les traitements de l'application. Il s'appuie sur les diagrammes de cas d'utilisation, de séquences et de classes. Pour construire ces diagrammes, nous avons adopté les étapes suivantes :

Après l'identification des différents acteurs, ainsi que de différentes fonctions du système à concevoir durant la partie d'analyse, nous mettons en évidence :

- Les diagrammes de cas d'utilisation mis en œuvre par les différents acteurs du système.
- Les diagrammes de séquence, on formalise graphiquement le ou les scénarios qui décrivent chaque cas d'utilisation.
- Le diagramme de classes, on formalise les classes.

Finalement, la démarche globale que nous avons adoptée pour la modélisation de l'aspect dynamique de notre application peut être résumée à travers les étapes suivantes :

- Identification et représentation des cas d'utilisation.
- Elaboration des diagrammes de séquences.
- Elaboration des diagrammes de classes.

3.1. Représentation des diagrammes de cas d'utilisation

3.1.1. Définition d'un cas d'utilisation

Un cas d'utilisation précise le comportement d'un système ou d'une partie d'un système et décrit un ensemble de séquences d'actions. Les cas d'utilisation servent à saisir le comportement attendu d'un système en cours de développement, sans avoir à préciser la façon dont ce comportement est réalisé. En UML, tous les comportements sont modélisés sous la forme de cas d'utilisation via un diagramme de cas d'utilisation.

3.1.2. Identification des cas d'utilisation

Acteur	Cas d'utilisation	Sous cas d'utilisation
Administrateur	**S'authentifier**	
	Gestion des comptes	Ajouter un compte. Modifier un compte. Supprimer un compte.

Utilisateur	**S'authentifier**	
	Caisse	Facture initiale. Facture avoir. Facture complémentaire. Facture pro format.
	Recherche multicritères	Numéro de la facture Numéro du reglement Date de la facture Date du reglement Numéro du dossier Entre deux dates
	Suivi règlement des factures à crédit.	Facture initiale. Facture complémentaire.
	Réédition	Facture (initiale, avoir, complémentaire). Bon de reçu. Etat de caisse. Historique d'une facture.

Tableau I.1 : Représentation des cas d'utilisation

3.1.3. Définition du diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation sont des diagrammes UML utilisés pour donner une vision globale du comportement fonctionnel d'un système logiciel. Ils sont utiles pour des présentations auprès de la direction ou des acteurs d'un projet, mais pour le développement, les cas d'utilisation sont plus appropriés.

- **La relation « include » :**

  Le cas d'utilisation combine précisément et de manière obligatoire un autre cas d'utilisation à l'endroit spécifie.

- **La relation « extend » :**

  Le cas d'utilisation combine tacitement et de manière facultative un autre cas d'utilisation à l'endroit spécifie.

3.1.4. Diagramme de cas d'utilisation

a. diagramme de cas utilisation « Administrateur »

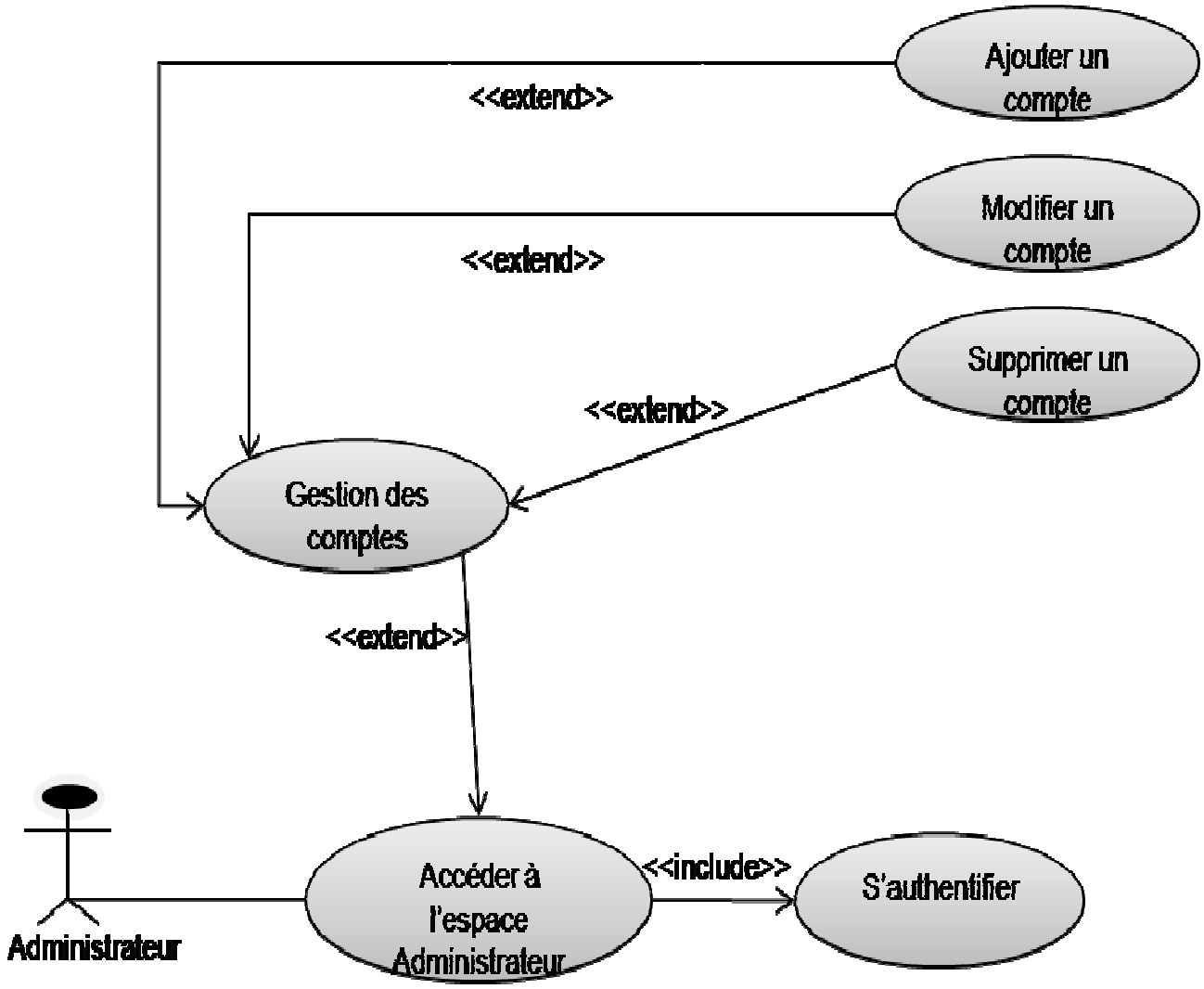


Figure I.2 : Diagramme de cas d'utilisation « «Administrateur ».

b. diagramme de cas d'utilisation « Utilisateur »

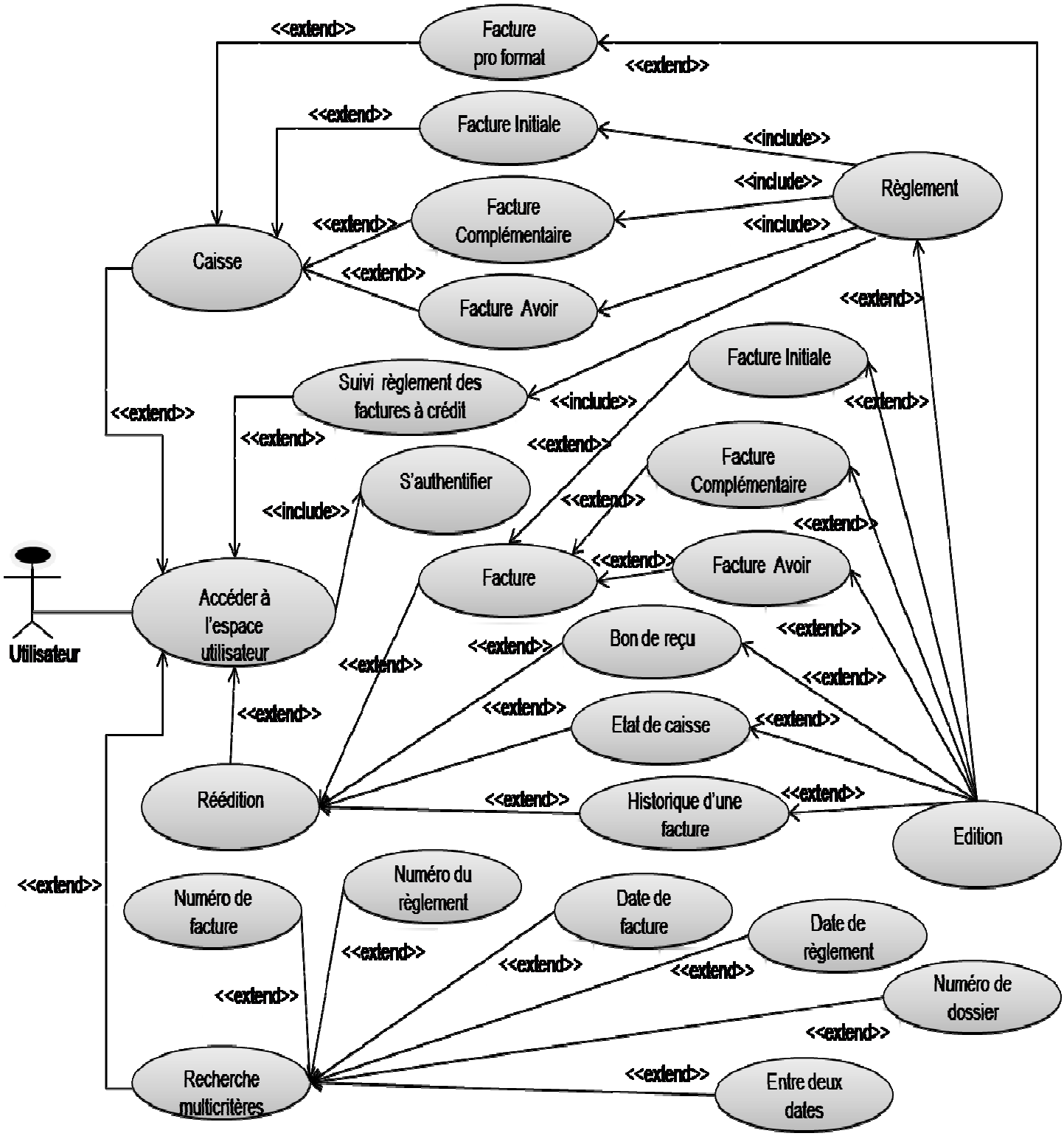


Figure I.3: Diagramme de cas d'utilisation « Utilisateur ».

3.2. Représentation des diagrammes de séquence

3.2.1. Définition du diagramme de séquence

Indique l'interaction entre plusieurs partenaires de communication, également appelés lignes de vie. Les principales informations contenues dans les diagrammes de séquences sont les messages échangés entre les lignes de vie. Un diagramme de séquences met toujours l'accent sur l'ordre chronologique des messages.

Ces diagrammes peuvent être utilisés pour modéliser les responsabilités et les collaborations sans prendre en compte les mécanismes définis par l'architecture du système.

3.2.2. Les diagrammes de séquence

Nous allons présenter dans ce qu'il suit les diagrammes de séquence pour quelques cas d'utilisation, mais avant tout, nous définirons quelques notions utilisées.

Quelques définitions :

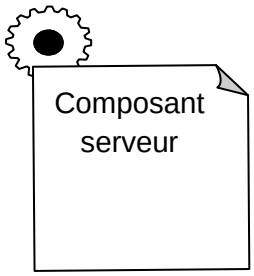
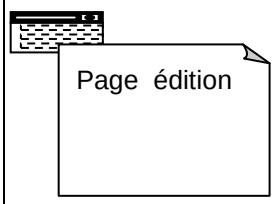
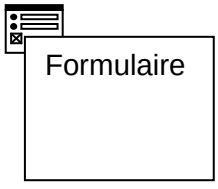
Composant	Description	icone
Composant serveur	Un composant serveur représente une classe qui possède des scripts exécutés par le serveur. Ces scripts interagissent avec des ressources serveur tels que les bases de données ou systèmes externes. Les opérations de l'objet représentent des fonctions dans les scripts et ces attributs représentent les variables qui sont visibles dans la portée de la page.	
Page client	Une instance d'une page client est une page Web formatée en langage Web. Les pages clients qui sont restituées par des navigateurs client, peuvent contenir des scripts interprétés par les navigateurs. Les fonctions d'une page client correspondent aux fonctions des scripts de la page Web.	
Page client	Une classe formulaire est un ensemble de champs de saisie faisant partie d'une page client. Les attributs de cette classe correspondent aux éléments de saisie d'un formulaire Web (zone de saisie, zone de texte, boutons d'option à cocher et éléments cachés).	

Tableau I.2 : Présentation des éléments de diagramme de séquence

a- Diagramme de séquence pour le cas d'utilisation « modifier un compte » :

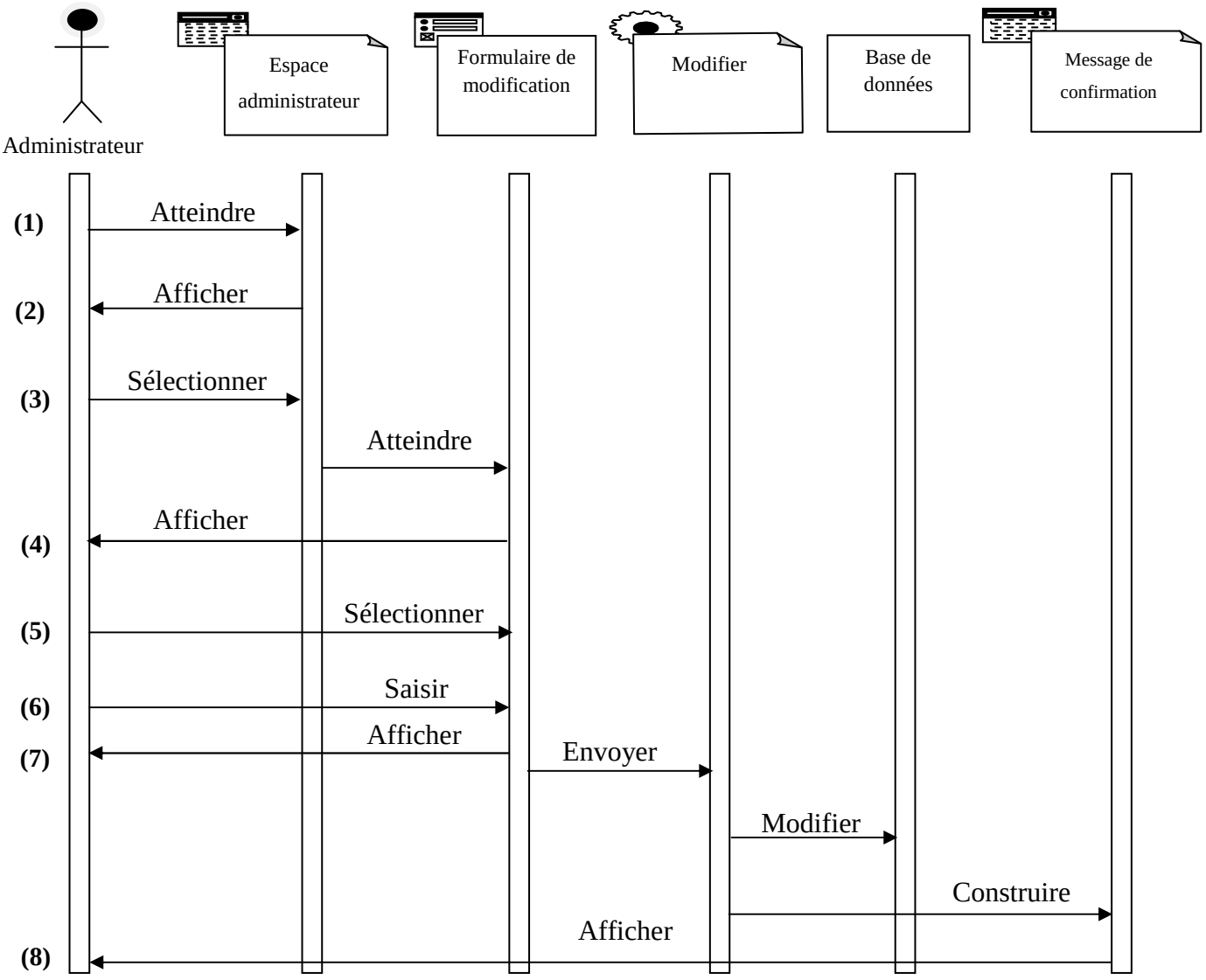


Figure I.4: Diagramme de séquence «Modifier un compte utilisateur».

1) L'administrateur atteint son espace.

2) L'administrateur visualise son espace.

3) L'administrateur sélectionne la modification d'un utilisateur.

4) Le système retourne le formulaire « Modifier un utilisateur ».

5) L'administrateur sélectionne l'ancien Identifiant.

6) L'administrateur saisit les nouvelles coordonnées et envoyer les informations.

7) Le système affiche un message en cas d'erreur.

8) Le système affiche un message de confirmation ou d'erreur.

b- Diagramme de séquence pour le cas d'utilisation « facture initiale » :

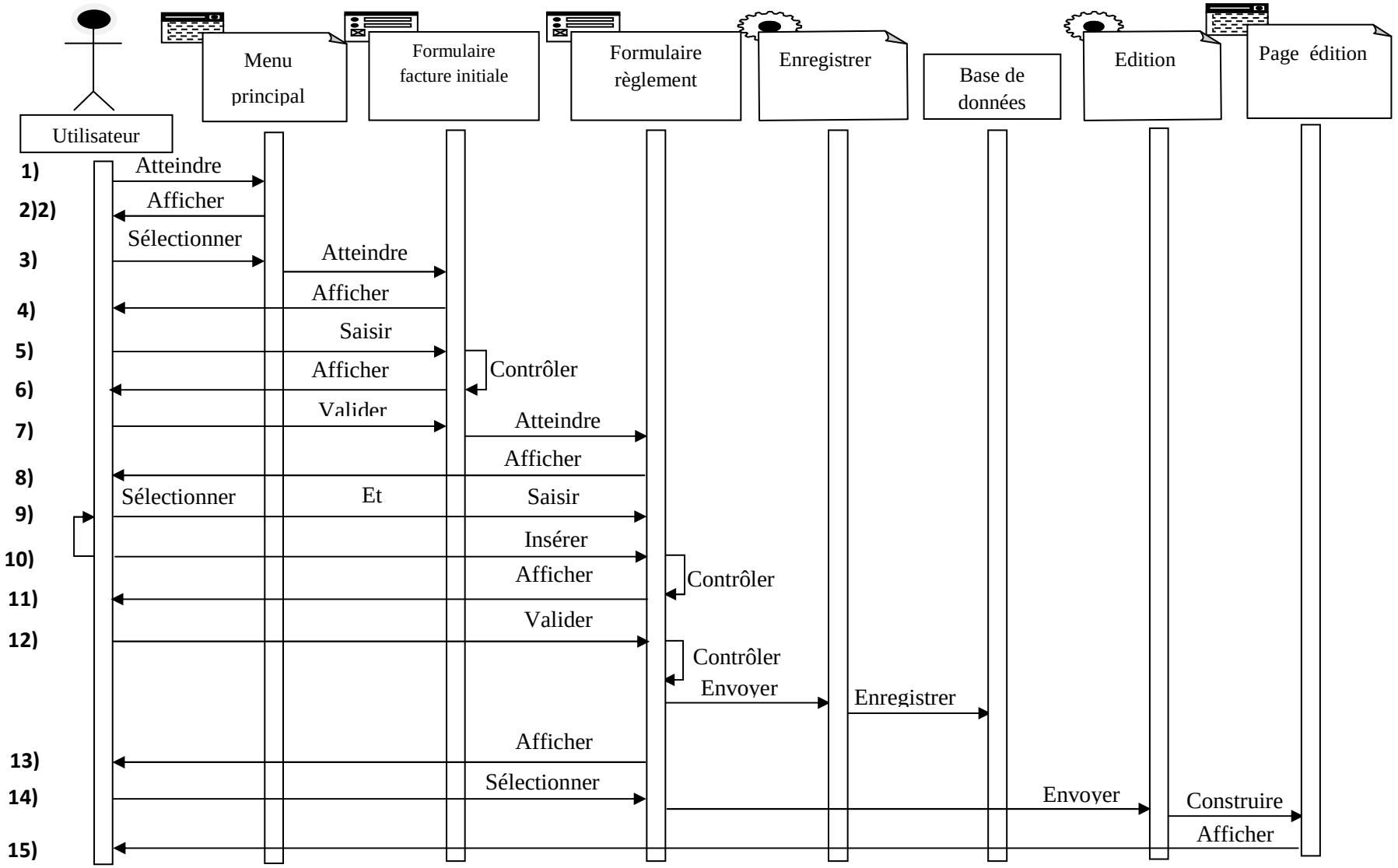


Figure I.5: Diagramme de séquence : «Facture initiale »

1) L'utilisateur atteint le menu principal.
2) Le système affiche le menu principal.
3) l'utilisateur sélectionne « Facture Initiale ».
4) Le système affiche le formulaire Facture Initiale.
5) L'utilisateur saisit le numéro du dossier de l'affaire à facturer.
6) Le système affiche les éléments du dossier recherché.
7) L'utilisateur valide la Facture Initiale.
8) Le système affiche le formulaire règlement.
9) L'utilisateur sélectionne et saisit les informations nécessaires : mode de règlement, montant de la tranche, l'émetteur et l'adresse banque.
10) L'utilisateur ajoute des tranches, autant de fois qu'il veut, tant que le montant restant n'est pas égal à zéro (si le client n'est pas conventionné).
11) Le système affiche le montant réglé et le montant restant.
12) L'utilisateur valide le règlement (enregistrement de la facture initiale et son règlement si il existe).
13) Le système affiche toutes les informations inhibées.
14) L'utilisateur sélectionne le bouton « imprimer ».
15) Le système affiche l'état de sortie de la facture initiale.

c- Diagramme de séquence pour le cas d'utilisation « facture avoir » :

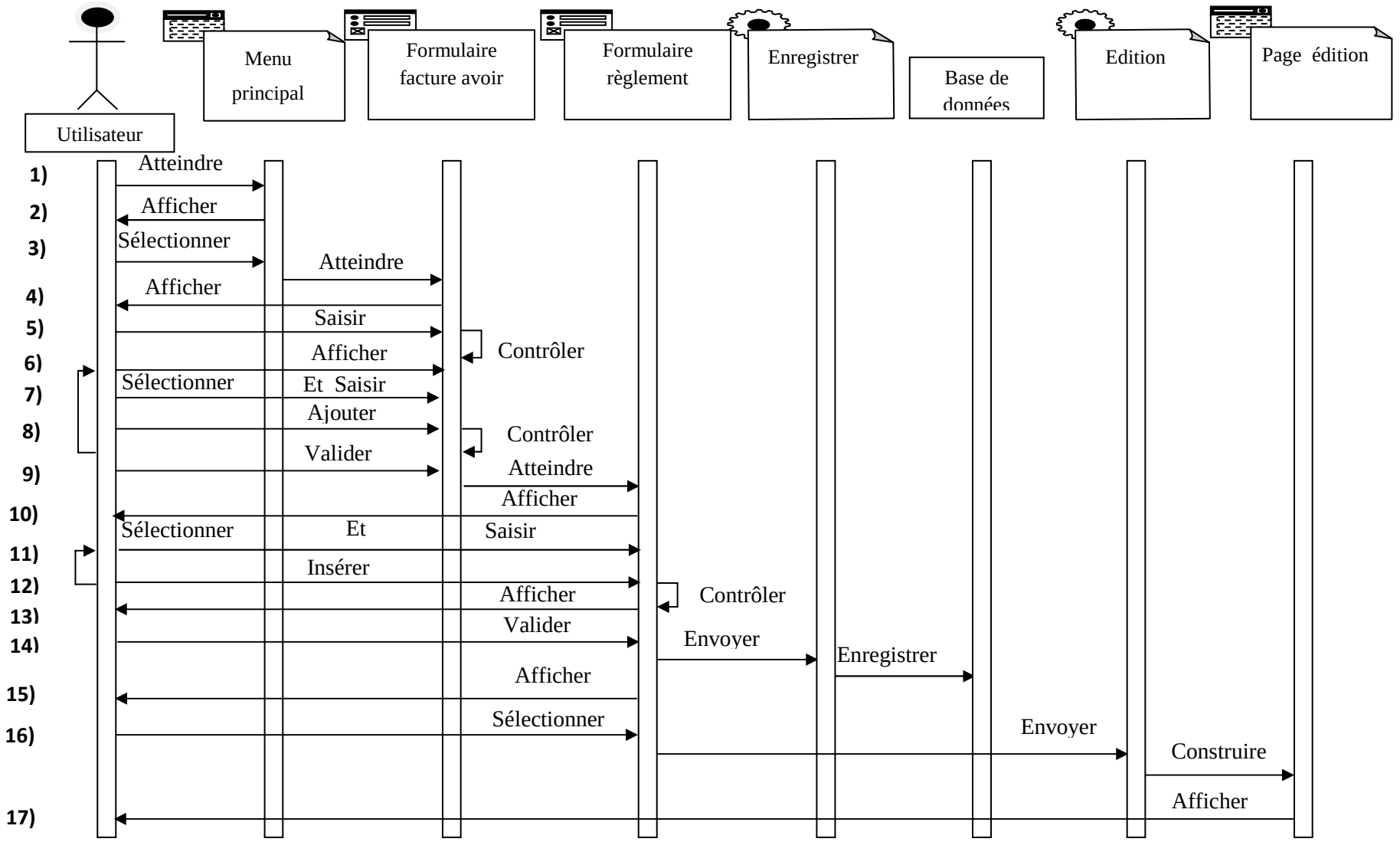


Figure I.6: Diagramme de séquence «Facture avoir ».

1) L'utilisateur atteint le menu principal.
2) Le système affiche le menu principal.
3) l'utilisateur sélectionne « Facture Avoir ».
4) Le système affiche le formulaire Facture Avoir.
5) L'utilisateur saisit le numéro de la facture initiale.
6) Le système affiche les rubriques de la facture initiale avec leurs montants.
7) L'utilisateur sélectionne la rubrique voulue et saisit le nouveau montant de la rubrique.
8) L'utilisateur ajoute la nouvelle rubrique.
9) L'utilisateur valide la facture avoir.
10) Le système affiche le formulaire règlement.
11) L'utilisateur sélectionne le mode de règlement de la tranche et saisit son montant.
12) L'utilisateur ajoute la tranche.
13) Le système affiche le montant réglé et le montant restant.
14) L'utilisateur valide le règlement.
15) Le système affiche toutes les informations inhibées.
16) L'utilisateur sélectionne le bouton « imprimer ».
17) Le système affiche l'état de sortie de la facture avoir.

d- Diagramme de séquence pour le cas d'utilisation « suivi règlement » :

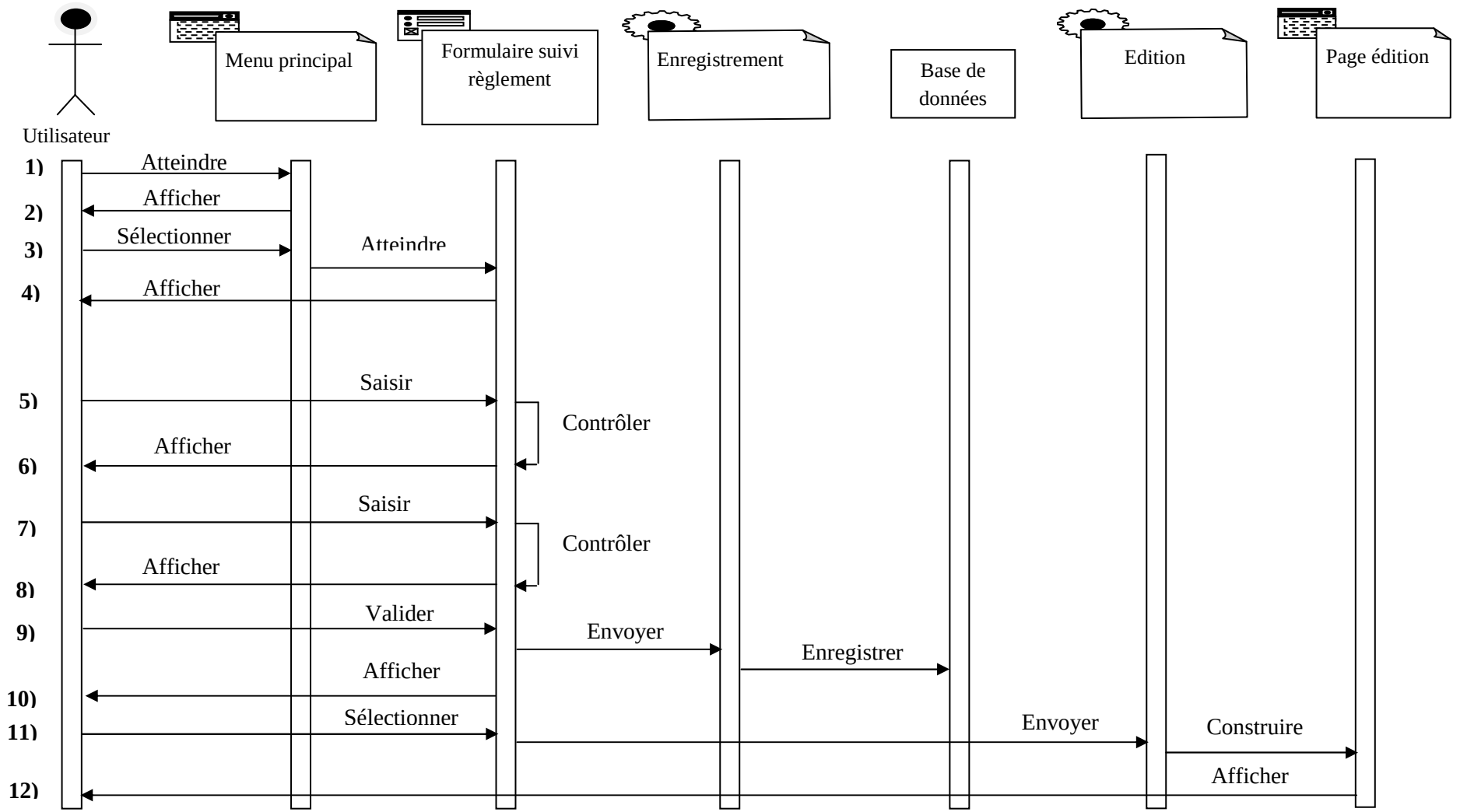


Figure I.7: Diagramme de séquence : «Suivi règlement ».

1) L'utilisateur atteint le menu principal.
2) Le système affiche le menu principal.
3) L'utilisateur sélectionne « suivi règlement », pour régler la facture à crédit (client conventionné).
4) Le système affiche le formulaire Suivi règlement.
5) L'utilisateur saisit le numéro de la facture non réglée.
6) Le système affiche les éléments de la facture recherché et le montant restant.
7) L'utilisateur saisit les informations de chaque tranche du règlement et son type.
8) Le système affiche le montant réglé et le montant restant de chaque tranche.
9) L'utilisateur valide le règlement.
10) Le système affiche toutes les informations inhibées.
11) L'utilisateur sélectionne le bouton « imprimer ».
12) Le système affiche l'état de sortie du bon de reçu.

e- Diagramme de séquence pour le cas d'utilisation « état de caisse » :

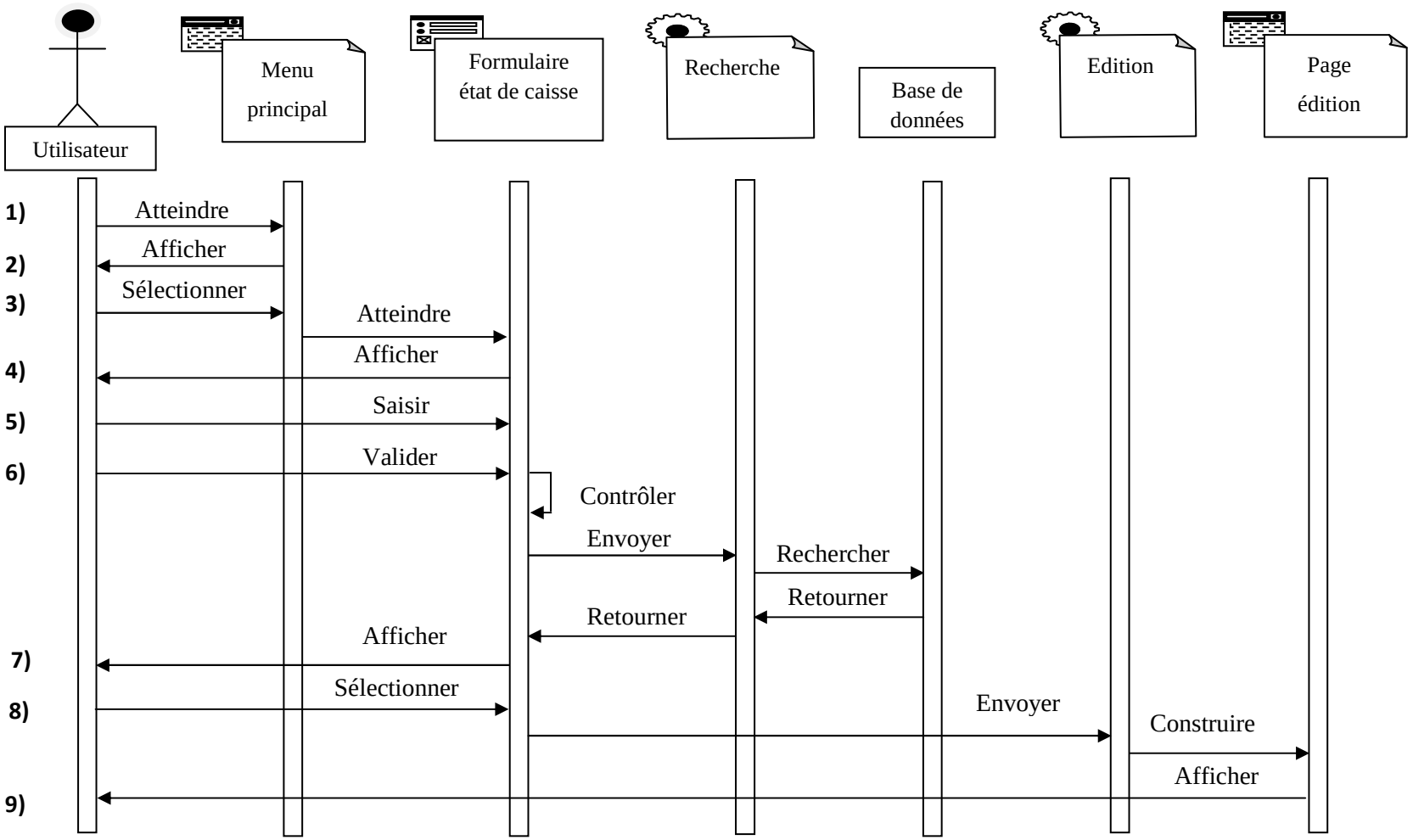


Figure I.8: Diagramme de séquence «Etat de caisse ».

1) L'utilisateur atteint le menu principal.
2) Le système affiche le menu principal.
3) l'utilisateur sélectionne « Etat de caisse ».
4) Le système affiche le formulaire Etat de caisse.
5) L'utilisateur saisit les deux dates d'entrée.
6) L'utilisateur valide le formulaire.
7) Le système affiche les résultats.
8) L'utilisateur sélectionne le bouton « imprimer ».
9) Le système affiche l'état de sortie d'état de caisse.

f- Diagramme de séquence pour le cas d'utilisation « recherche multicritères » :

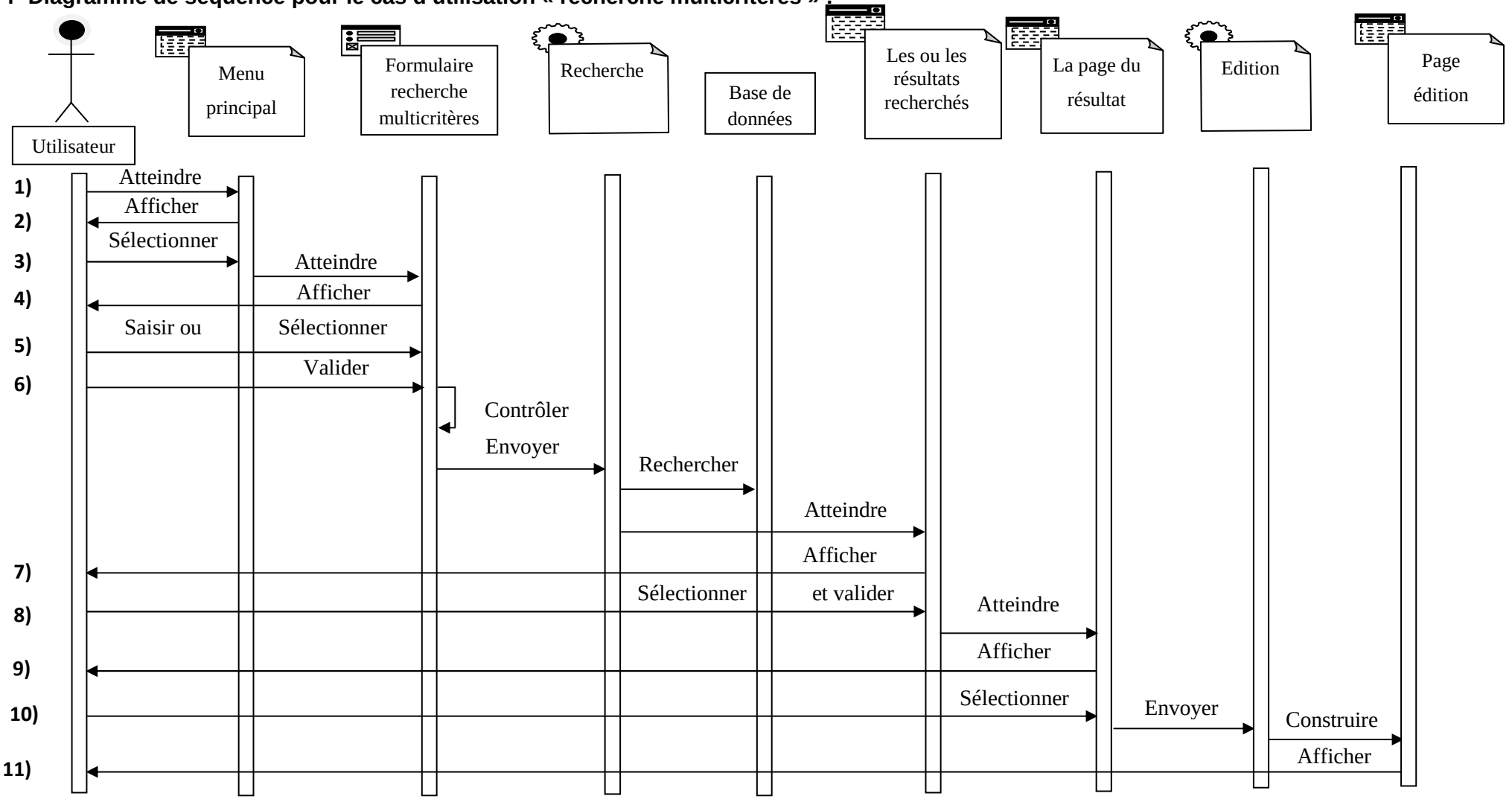


Figure I.9: Diagramme de séquence «Recherche multicritères ».

1) L'utilisateur atteint le menu principal.
2) Le système affiche le menu principal.
3) l'utilisateur sélectionne « Recherche multicritères».
4) Le système affiche le formulaire Recherche multicritères.
5) L'utilisateur saisit le critère ou les critères voulus.
6) L'utilisateur valide le formulaire.
7) Le système affiche le ou les résultats retrouvés.
8) L'utilisateur sélectionne un résultat et valide.
9) Le système affiche les informations du résultat sélectionné.
10) L'utilisateur sélectionne le bouton « imprimer ».
11) Le système affiche l'état de sortie du résultat.

3. 3. Représentation des diagrammes de classes

3.3.1. Définition du diagramme de classes

Le diagramme de classes constitue un élément très important de la modélisation : il permet de définir quelles seront les composantes du système final : il ne permet en revanche pas de définir le nombre et l'état des instances individuelles.

Définition et représentation des éléments du diagramme de classes

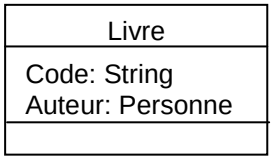
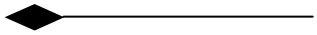
Classes	Les classes sont les modules de base de la programmation orientée objet.	
Composition	Un lien de composition symbolise l'existence d'une liaison particulière, dite 'forte', entre deux entités (classes).	

Tableau I.3 : Présentation des éléments de diagramme de classes

3.3.3. Le diagramme de classes

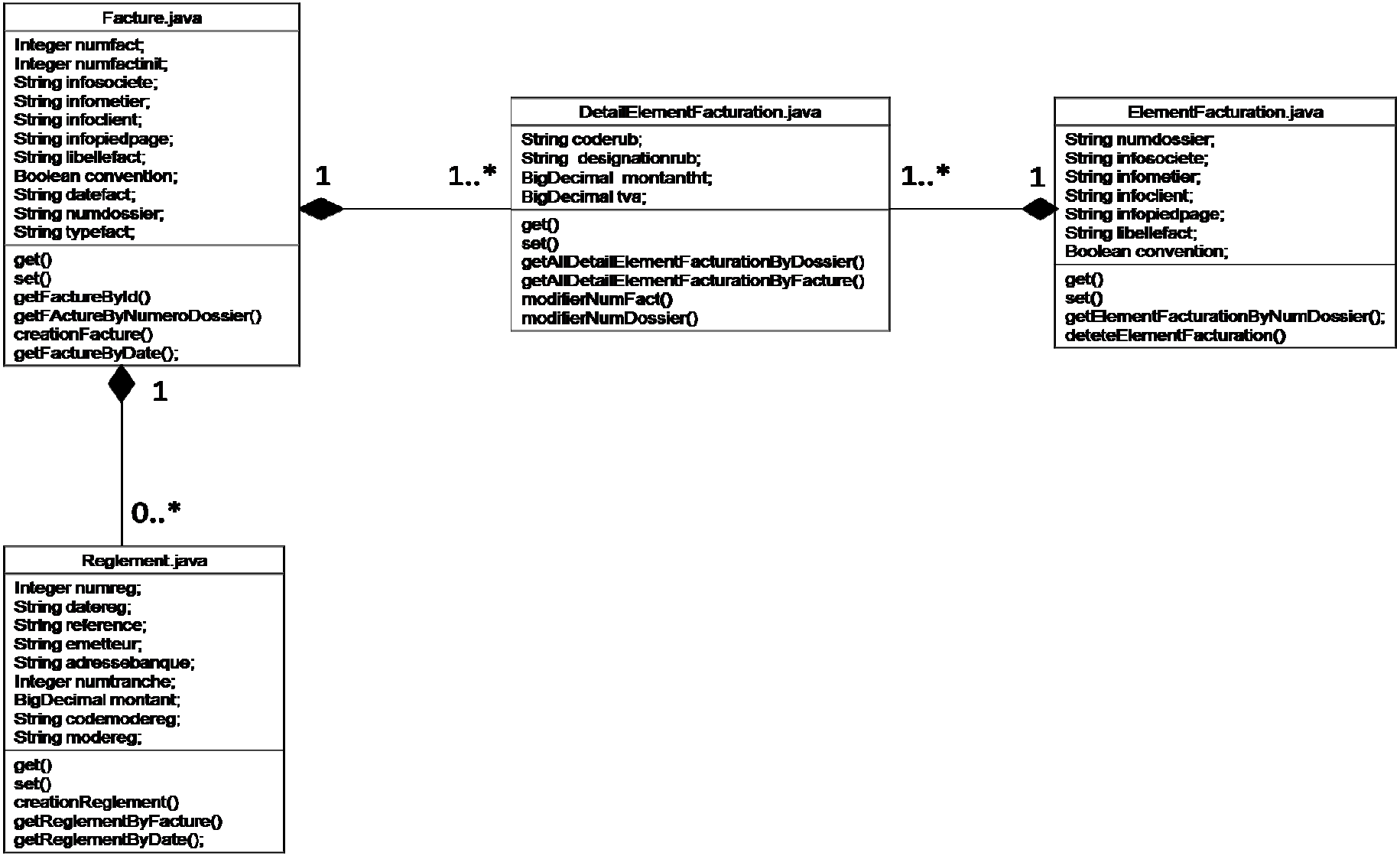


Figure I.10 : Diagramme de classes.

4 .Modélisation de l'aspect statique de la solution proposée

Le processus de modélisation de l'aspect statique se résume par les étapes suivantes:

- Présenter les règles de gestion.
- Construire le dictionnaire de données.
- Construire le modèle Entité-Association.
- Transformer le schéma entité-association en modèle Relationnel.

4.1. Les règles de gestion

- Un dossier contient au moins une rubrique.
- Une facture contient au moins une rubrique.
- Une fois un dossier est facturé, le système le supprime automatiquement.
- Une facture peut avoir un ou plusieurs reglement.

4.2. Le dictionnaire des données

Un dictionnaire des données est une collection de métadonnées ou de données de référence nécessaire à la conception d'une base de données.

Nom	Désignation	Entité/association	Type	Identité
code_rub	Code de la rubrique	detail_element_facturation	Alphanumérique	Oui
designation_rub	Désignation de la rubrique	detail_element_facturation	Alphanumérique	Non
montant_ht	Montant hors taxe	detail_element_facturation	Numérique	Non
tva	Taux des valeurs acquises	detail_element_facturation	Numérique	Non
num_dossier	Numéro du dossier	element_facturation	Alphanumérique	Oui
num_dossier	Numéro du dossier	facture	Alphanumérique	Non
info_societe	Informations de la société	element_facturation facture	Alphanumérique	Non

info_metier	Informations du métier	element_facturation facture	Alphanumérique	Non
info_client	Informations du client	element_facturation facture	Alphanumérique	Non
info_pied _page	Informations du pied de page	element_facturation facture	Alphanumérique	Non
convention	Convention	element_facturation facture	Booléen	Non
libelle_fact	Libelle de la facture	element_facturation facture	Alphanumérique	Non
num_fact	Numéro de la facture	facture	Numérique	Oui
num_fact_init	Numéro de la facture initiale	facture	Numérique	Non
type_fact	Type de la facture	facture	Alphabétique	Non
date_fact	Date de la facture	facture	Numérique	Non
num_reg	Numéro du reglement	Reglement	Numérique	Oui
date_reg	Date du reglement	Reglement	Numérique	Non
reference_reg	Référence du reglement	Reglement	Alphanumérique	Non
montant	Montant du reglement	Reglement	Numérique	Non
code_mode_reg	Code du mode de reglement	Reglement	Alphanumérique	Non
mode_reg	Mode de reglement	Reglement	Alphanumérique	Non

num_tranche	Numéro de la tranche	Reglement	Numérique	Non
emetteur	Emetteur	Reglement	Alphanumérique	Non
adresse_banque	Adresse de la banque	Reglement	Alphanumérique	non

Tableau I.4 : Dictionnaire des données

4.3. Le modèle Entité-Association

Le modèle entité-association est l'un des modèles conceptuels les plus utilisés pour construire une base de données.UML fournit une notation graphique très simple qui facilite la modélisation de données.

Le modèle entité-association est généralement fondé sur quatre principaux concepts (l'entité[1], l'association[2], la propriété[3], et les cardinalités [4]) qui permettent de décrire un ensemble de données relatives à un domaine défini. Le modèle Entité-Association de notre cas est comme suit :

[1] Un objet, une chose concrète ou abstraite qui peut être reconnue distinctement.

[2] Un lien entre plusieurs entités.

[3] Caractéristique de description d'une entité ou d'une association.

[4]

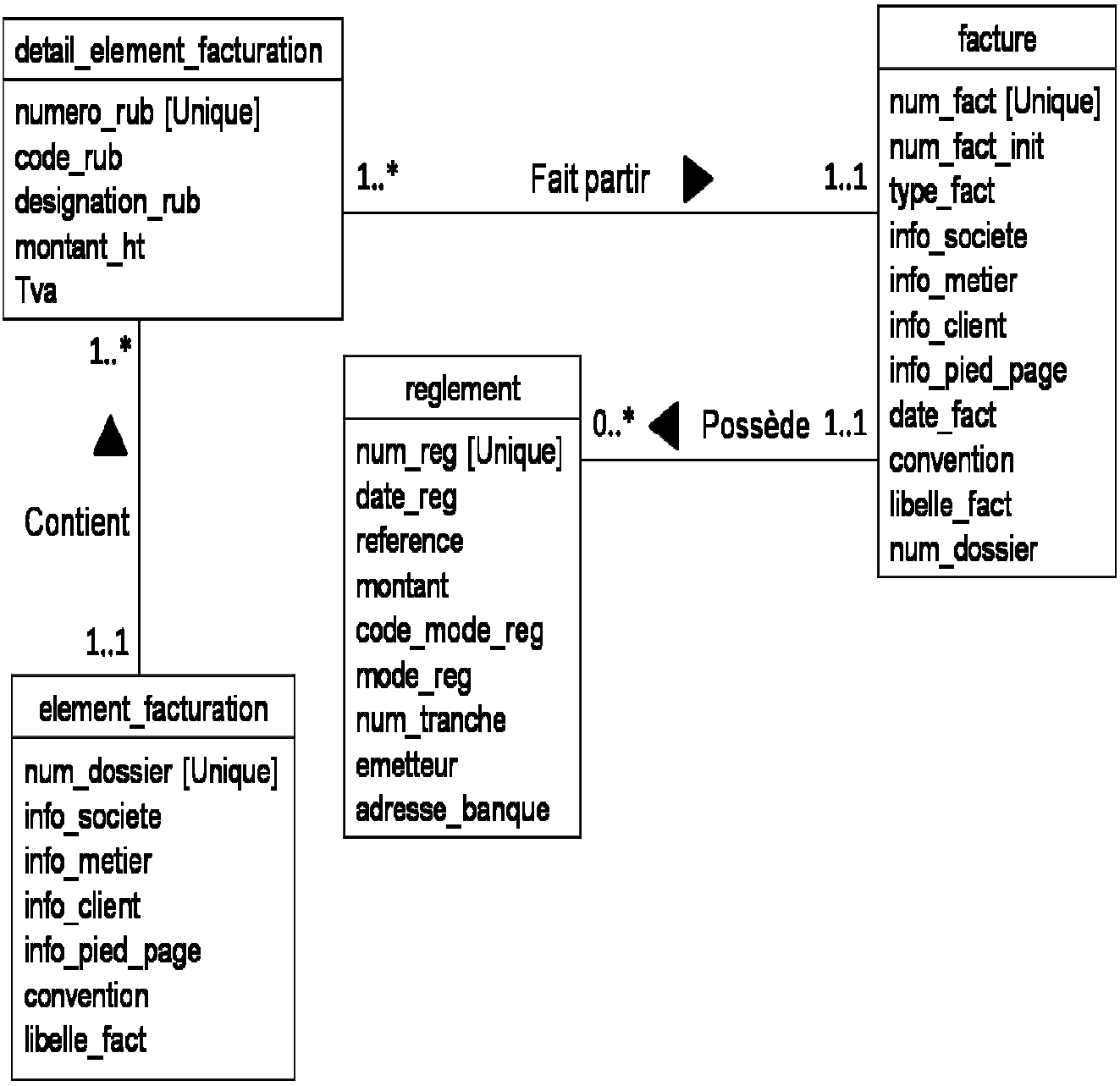


Figure I.11: Modèle Entité-Association.

4.4. Le modèle Relationnel

C'est un modèle de données découlant d'un modèle conceptuel mais qui le raffine pour tenir compte des caractéristiques du type de SGBD utilisé pour la réalisation de la BD (Base de Données). La figure suivante présente le modèle relationnel de notre cas :

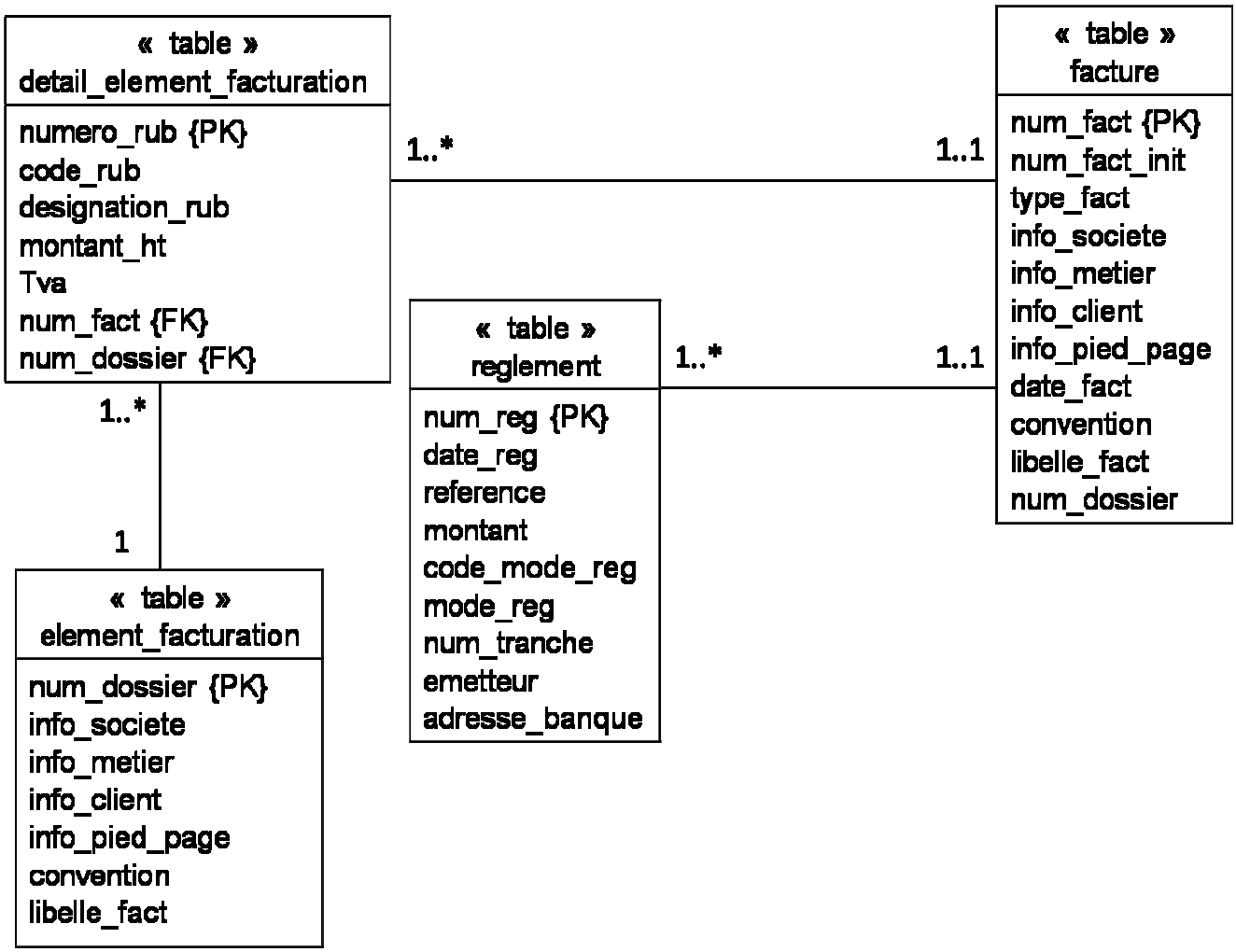


Figure I.12: Modèle Relationnel.

Remarque :

PK: *Primary Key*, c'est-à-dire clé primaire.

FK: *Foreign Key,* c'est-à-dire clé étrangère.

Conclusion

Tout au long de ce chapitre, dans la section analyse, nous avons présenté les différents besoins auxquels le système doit répondre. Dans la section conception, en utilisant le langage UML, nous avons modélisé les différents cas du futur système identifié.

Ce stade de développement auquel nous sommes arrivés, nous permet aisément de mettre en œuvre l'application, en utilisant des techniques de développement qui feront l'objet du chapitre 1 de la partie suivante.

Partie III

Réalisation

Chapitre I : Architectures et outils de développement

- ¨Architectures
- Développement
- Serveur de données
- Serveur d’applications
- Environnement de développement

Chapitre II : Architecture de l’application

- Schéma fonctionnel de l’application
- Diagramme de classes de l’application
- Schéma applicatif de l’application
- Schéma applicatif de l’application
- Schéma applicatif détaillé de l’application
- Principe de fonctionnement de l’application
- Présentation de quelques interfaces

Pour la réalisation de notre application nous avons eu recours à plusieurs architectures et outils de développement, nous les citons dans ce qui suit.

1. Architectures

1.1. Java EE (Java Entreprise Edition)

La plate-forme Java EE s'appuie entièrement sur le langage Java. Java EE est donc une norme, qui permet à des développeurs de réaliser leur propre application qui implémente en totalité ou partiellement les spécifications de SUN. En simplifiant, il est possible de représenter Java EE comme un ensemble de spécifications d'APIs (ensemble de librairies et de services), une architecture et une méthode de packaging.

Il existe actuellement beaucoup d'autres plates-formes de développement qui sont basées sur d'autres langages (C#, PHP5, .NET ...). Les principaux avantages d'utiliser Java EE (est donc Java) sont la portabilité, l'indépendance, la sécurité et la multitude de librairies proposées. Le développement d'applications d'entreprise nécessite la mise en œuvre d'une infrastructure importante. Beaucoup de fonctionnalités sont utilisées et développées, le but étant de produire des applications sures, robustes et faciles à maintenir. Certains services sont d'ailleurs récurrents comme : l'accès aux bases données, l'envoi de mails, les transactions, la gestion de fichiers, la gestion d'images, le téléchargement, le chargement et la supervision du système...etc.

C'est pour cela que l'architecture Java EE est intéressante car tous les éléments fondamentaux sont déjà en place. Pas besoin de concevoir une architecture, des librairies et des outils spécialement adaptés. Cela nécessite un temps et un investissement considérables.

Enfin, la plate-forme Java EE est basée sur des spécifications, ce qui signifie que les projets sont portables sur n'importe quel serveur d'application conforme (Tomcat, JBoss, WebSphere et Weblogic...) à ces spécifications. Cette implémentation est gratuite et permet de bénéficier de la totalité de l'API sans investissement. La plate-forme Java EE est la plus riche des plates-formes Java et offre un environnement standard de développement et d'exécution d'applications d'entreprise multi-tiers.

Le fait que Java EE soit standardisé, il a contribué à son adoption par de très nombreux éditeurs de logiciels/outils informatique. Ces éditeurs associés à Sun Microsystems font partis du JCP (Java community Process). Le Java Community Process regroupe les entreprises suivantes : Sun, IBM, Oracle, Borland, Nokia, Sony, la fondation Apache, ObjectWeb... etc. (*Pour plus de détails voir l'annexe*).

1.2. Design Pattern

Un Design Pattern est une solution à un problème récurrent dans la conception d'applications orientées objet. Un patron de conception décrit alors la solution éprouvée pour résoudre ce problème d'architecture de logiciel.

Le concepteur objet tente de faciliter la réutilisation et la maintenance du code. On peut donc concevoir un modèle d'application comme une forme d'organisation transposable à plusieurs applications. Ces systèmes peuvent apparaître complexes aux débutants, voire inutiles, il est pourtant très important d'en connaître plusieurs et de les appliquer.

A noter qu'en se plaçant au niveau de la conception les Design Patterns sont indépendants des langages de programmation utilisés. (*Pour plus de détails voir l'annexe*).

2. Développement

2.1. JAVA

C'est un langage de programmation orienté objet, développé par Sun Microsystems. Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris). Java donne aussi la possibilité de développer des programmes pour téléphones portables et assistants personnels. Enfin, ce langage peut-être utilisé sur internet pour des petites applications intégrées à la page web ou encore comme langage serveur .

2.2. HTML (Hypertext Markup Language)

HTML est un langage dit de «« marquage » ou de « balisage », est le format de données conçu pour représenter les pages web. C'est un langage de balisage qui permet d'écrire de l'hypertexte, d'où son nom.

HTML permet également de structurer sémantiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des éléments programmables tels que des applets. Il est souvent utilisé conjointement avec des langages de programmation (JavaScript) et des formats de présentation (feuilles de style en cascade).

Le langage HTML est un langage qui peut être lue par des ordinateurs de différentes marques et aussi par des navigateurs divers.

2.3. JavaScript

JavaScript est langage de script, qui permet notamment de manipuler les éléments d'une page Web, d'interagir avec le navigateur internet et de réagir aux actions de l'utilisateur. Contrairement à certaines idées reçus, JavaScript est un langage moderne qui offre de nombreuses fonctionnalités puissantes comme :

- Le développent orienté objet ou encore la gestion des exceptions.
- Les fonctions du JavaScript ne doivent pas attendre pour des réponses de leurs serveurs pour agir, ce qui accélère l'ouverture des sites web.

Le JavaScript est relativement simple et facile à apprendre. Il ne nécessite pas un programme spécial pour l'interpréter, ni pour l'écrire. De plus, JavaScript n'occupe pas un grand espace sur les sites web.

2.4. CSS (Cascading Style Sheets)

CSS (feuilles de styles en cascade), est un langage des styles, sert à personnaliser la présentation d'un document. Les feuilles de style en cascade CSS en particulier améliore l'apparence et la structure des documents HTM.

Grâce aux CSS, on peut surtout définir pour une partie du texte ou pour tout un document :

- Les polices de caractères (type, taille, style), les couleurs, la présentation des images, des tableaux, les alignements, etc.

2.5. JSF 1.2 (Java Server Faces)

JSF est un Framework fait en JAVA faisant partie du standard J2EE, il offre au développeur une bibliothèque d'outils très large. D'autre part, il permet de mieux organiser le code, c'est à dire de séparer la vue (code html, JavaScript et CSS) du contrôleur (code java pur, lié aux fonctionnalités de l'application web) et du modèle (données et connexion à la base de données).

JSF a deux avantages principaux :

- Un développement rapide grâce à ses composants.
- Une maintenance simplifiée grâce à sa structure.

JSF permet :

- Une séparation nette entre la couche présentation et les autres couches d'une application web.
- Une mise en place d'un mapping HTML/OBJET.
- La un modèle riche de composants graphiques réutilisables.
- Une liaison simple entre les actions côté client de l'utilisateur et le code Java correspondant côté serveur.

Il existe plusieurs Frameworks Web Java dédiés au développement d'interfaces utilisateur mais aucun n'est un standard et va aussi loin que JSF.

2.6. RichFaces

Le projet RichFaces est une librairie de composants permettant l'intégration de comportement AJAX dans les vues, et ce, de manière très simple. Ce Framework s'appuie sur les composants JSF (Java Server Face). RichFaces permet aussi de définir des thèmes pour personnaliser simplement le rendu des pages.

2.7. JPQL (Java Persistance Query Language)

JPQL est un langage de requête orienté objet indépendant de la plateforme, défini dans la spécification Java Persistance API.

JPQL sert à exécuter des requêtes sur des entités persistées en base de données mais en travaillant sur les entités java correspondantes aux tables plutôt que sur les tables elles- mêmes.

2.8. BIRT (Business Intelligence Reporting Tools)

BIRT est une bibliothèque écrite en Java qui permet de générer des états à partir d'un modèle et d'une base de données ou (de listes) d'objets remplis. BIRT est accompagné d'un plugin Eclipse qui permet de dessiner les rapports à générer et les tester.

Remarque :

Le tableau ci-dessous résume, sous forme de point, les critères de choix de chaque outil de développement. L'ensemble des choix sont exigés par l'organisme d'accueil.

Outil	Critères de choix
Java	- Richesse : un des aspects important de l'environnement de JAVA est sa richesse de ses librairies des classes JAVA. - Interprète, portable et indépendant des architectures matérielles. Cette caractéristique est un avantage primordial pour Java face à des applications transmises par un réseau et exécutées sur des machines hétérogènes.
HTML	- Simplicité (il est simple à utiliser). - Indépendance (sa conception lui permet de rester indépendant vis à vis des plateformes et de pouvoir être échangé sur les réseaux). - Aucun logiciel spécialisé n'est nécessaire pour composer des pages HTML, il peut être composé sur n'importe quel système avec un simple éditeur de textes.
JavaScript	- Vitesse(Les fonctions du JavaScript ne doivent pas attendre pour des réponses de leurs serveurs pour agir).

	- Simplicité (le JavaScript est relativement simple et facile à apprendre). - Versatilité - Le JavaScript ne nécessite pas un programme spécial pour l'interpréter, ni pour l'écrire.
CSS	- Permettre des choses irréalisables en html. - Allègement du code-source des pages Web donc de plus petits fichiers et un chargement plus rapide. - plus facile de faire des changements d'ensemble (un seul fichier CSS à modifier plutôt que toutes les pages une à une)
JSF	- La rapidité (un développement rapide grâce à ses composants). La maintenabilité (une maintenance simplifiée grâce à sa structure). - Organisation du code (Il sépare la présentation des traitements).
RichFaces	- Elle est implémentée par le Framework JSF. - Permettre des choses irréalisables en html.
JPQL	- Un langage indépendant du SGBD (Système de gestion de base de données) car il se base sur l'approche orienté objet (Il permet de manipuler des objets, mais pas des tables). - Il se base sur le concept de Java EE.
BIRT	- Générez vos rapports dans le format de votre choix : Html, PDF, Word, Excel, PowerPoint, … - Utilisation une large palette de composants d'affichage pour composer des rapports : tableaux, grilles, images, graphiques … - Exploitation simultanée de différentes sources de données : bases de données relationnelles, fichiers XML, Web Services, objets Java, ...

Tableau I.1 : les causes de chois de outils de développement

3. Serveur de données

SQL SERVER 2005

SQL Server 2005 est un système de gestion de base de données relationnelle développée par Microsoft . Comme une base de données, il est un logiciel dont la fonction principale est de stocker, récupérer, traiter et sécuriser des données. Il fournit des accès contrôlés et des traitements de transactions rapides pour répondre aux besoins des applications les plus gourmandes en données utilisées au sein des entreprises. Il offre également les fonctions nécessaires pour faire face à des besoins de haute disponibilité.

4. Serveur d'applications

JBOSS 5.0.1

JBoss Application Server est un serveur d'applications J2EE Libre entièrement écrit en Java, publié sous licence GNU LGPL. Puisque le logiciel est écrit en Java, JBoss Application Server peut être utilisé sur tout système d'exploitation fournissant une machine virtuelle Java.

5. Environnement de développement

5.1. Eclipse 3.4 (Ganymede)

Eclipse IDE est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation.

Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

Conclusion

Dans ce chapitre nous avons présenté les architectures ainsi que les outils et l'environnement de développement utilisés pour la réalisation de notre application, mis en évidence les composants de la base de donnés.

Dans ce chapitre, nous commencerons par présenter le schéma fonctionnel de l'application, qui nous aidera à comprendre les fonctionnalités principales sur lesquelles nous nous sommes basés, ainsi que le déroulement de la navigation sur l'application. Ensuite nous allons passer au schéma applicatif qui a pour but de montrer les livrables que nous avons produit en respectant le principe de Java EE. Nous enchaînerons par la présentation du diagramme de classe et du principe de fonctionnement de notre application en se basant sur la facture initiale et son règlement et nous finirons par donner quelques interfaces de notre application.

1. Schéma fonctionnel de l'application

Le schéma fonctionnel ou l'arborescence d'une application, représente les différentes pages, organisées logiquement et hiérarchiquement.

L'arborescence d'une application est représentée sous la forme d'un arbre. La première page doit être la page d'accueil (appelée « la racine »), puis les autres pages apparaissent ensuite dans un ordre logique. L'arborescence d'une application aide l'internaute à comprendre la structure. La consultation et la mémorisation des pages est ainsi plus facile, rapide et efficace.

L'arborescence d'une application doit être au préalable réalisée sur papier avant de commencer la création. Elle est la base de toute application web à sa création.

Une fois l'utilisateur identifié grâce à son nom d'utilisateur et son mot de passe, le menu apparait et donne accès à 4 rubriques en principe, suivant le modèle que nous avons conçu (comme le schéma ci-dessus le montre). Mais dans la partie pratique de notre travail, nous nous sommes uniquement intéressés à réaliser la facture initiale (tous les autres : réédition, suivi des factures à crédit et recherche multicritères sont inhibés).

Dans ce module, l'utilisateur saisit des informations qui seront soumis à des contrôles, que nous détaillerons par la suite. Cela lui permettra de régler et éditer la facture initiale.

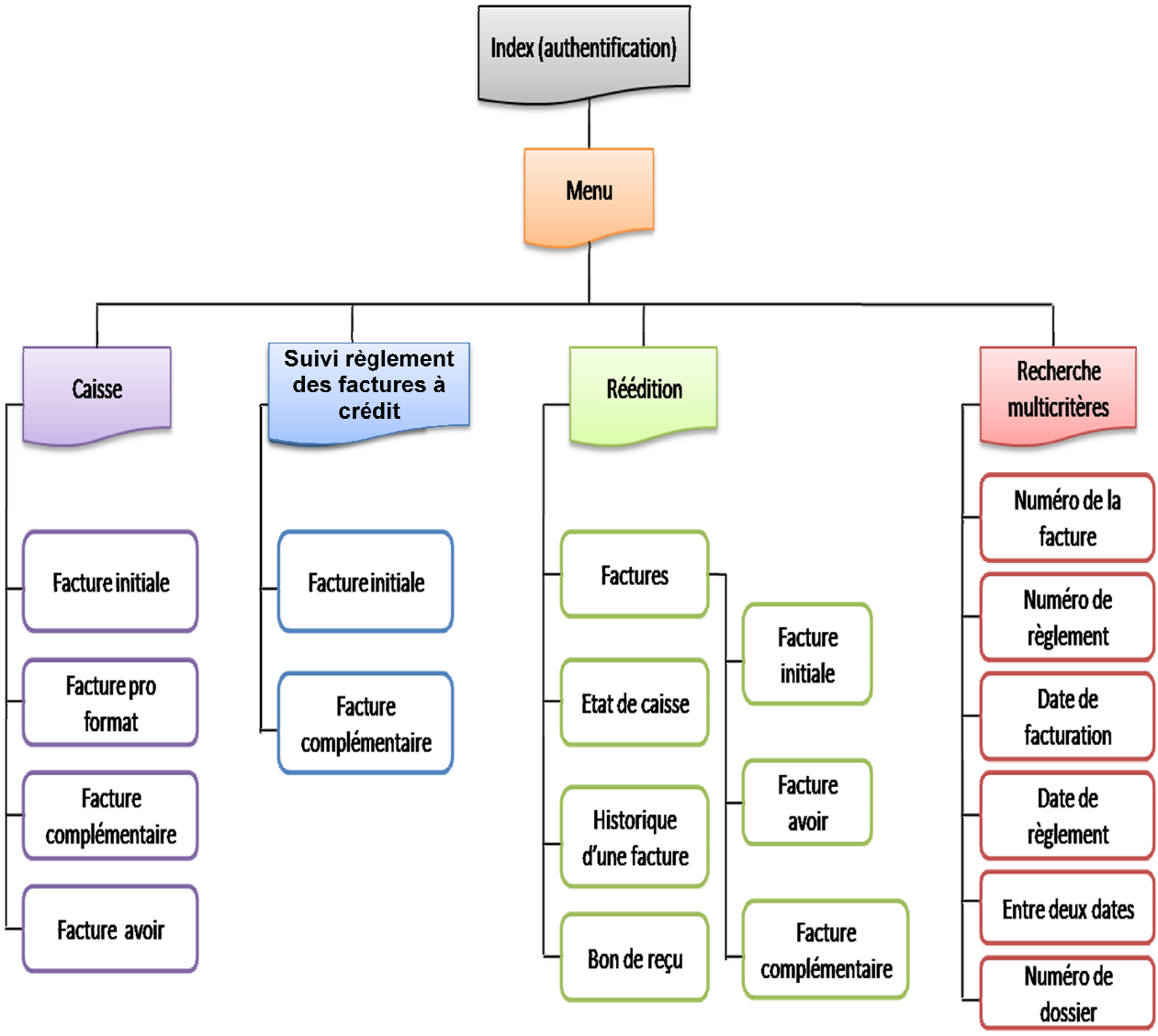


Figure II.1: Schéma fonctionnel de l'application.

2. Schéma applicatif de l'application

Développer une application Java EE revient à créer différents livrables, suivant la complexité des besoins de l'application.

Nous présentons ci-dessous le contenu de chaque livrable de notre application.

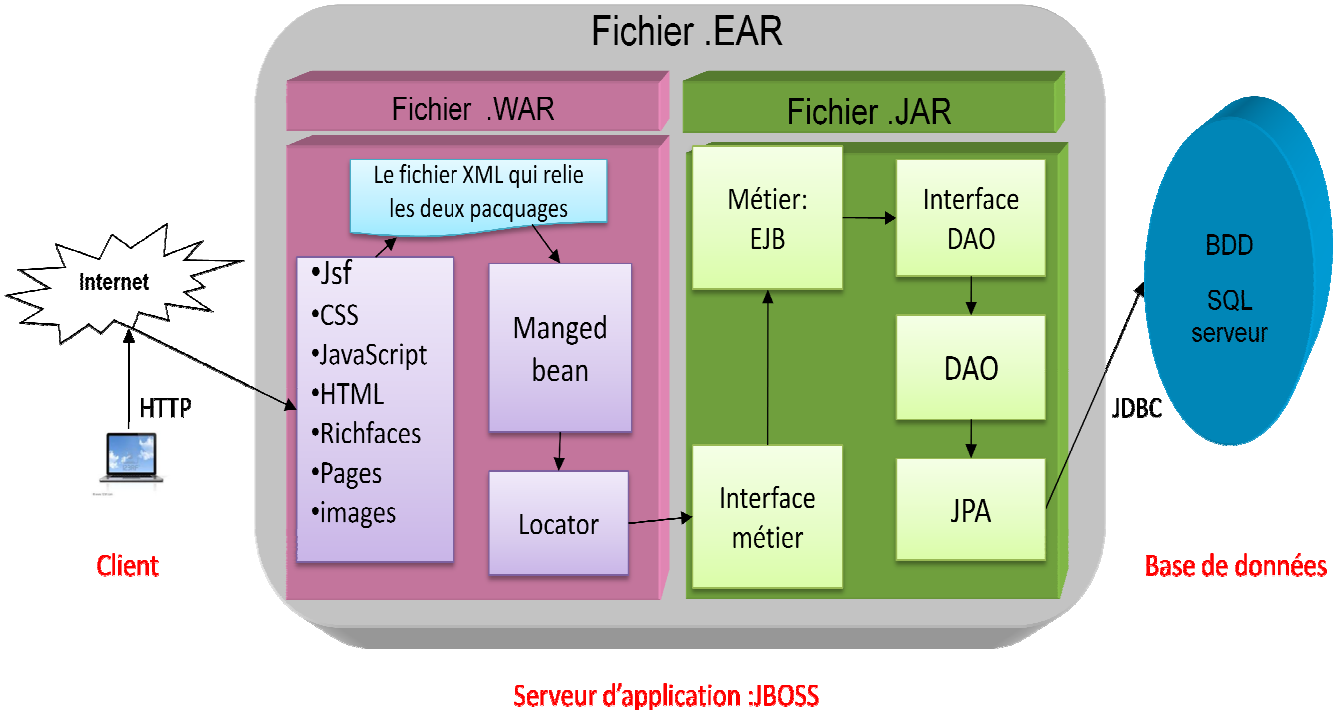


Figure II.2: Schéma applicatif de l'application.

a. Livrable EAR (Entreprise ARchive)

Comme son nom l'indique ce format de fichier est utilisé pour emballer tous les composants d'une application Web : les livrables d'extension war et jar.

Le but de ce format d'archive est de rendre le déploiement et à son tour le processus de maintenance, plus facile. Cela aura un fichier XML nommé web.xml comme le descripteur de déploiement. Ce descripteur de déploiement est utilisé par le conteneur Web pour déployer l'application web correctement.

Ce fichier doit être déployé sur le serveur d'application J2EE (JBoss).

b. Livrable WAR (Web ARchive)

C'est un fichier utilisé pour distribuer:

- Ecrans de l'application (pages, HTML, JSF).

- Images de l'application.
- Eléments de graphisme (Feuilles de style CSS, JavaScript, Richfaces).
- Classes Java métier (ManagedBean) : la fonction principale du ManagedBean est de sauvegarder des informations saisies dans le formulaire, ou récupérer depuis la base de données.
- Locator qui est une classe qui permet la communication entre la couche présentation et la couche métier, plus précisément entre le ManagedBean et couche métier.
- Fichiers de configuration permettant de configurer un service pour lequel la spécification J2EE n'est pas précise. On a utilisé deux fichiers de configuration : le fichier web.xml qui sert à configurer l'application web et le faces-config.xml qui contient tous les ManagedBean des formulaires.

c. Livrable JAR (Java ARchive)

Ce livrable contient :

- Les interfaces métier : Elles rendent les méthodes des classes métier visibles par les autres classes qui les utilisent (ManagedBean).
    - Elles représentent la frontière de communication entre les classes métier et la classe Locator.
- Les classes métier : ce sont les EJBs session, elles modélisent les processus métier de notre application (traitements).
- L'interface DAO : Elle permet de publier les méthodes de la classe DAO pour qu'elles soient utilisées par les classes métier.
- La classe DAO : c'est une classe qui implémente les quatre opérations de base pour la persistance des données (JAP), soient : **C**reate, **R**ead, **U**pdate, **D**elete.
- Les classes données (JPA) : ce sont les EJBs entity, elles représentent les données stockées dans la base de données.
- Un fichier de configuration Persistance.xml, permettant la connexion à la base de données.

3. Schéma applicatif détaillé de l'application

Dans cette partie, nous allons détailler le contenu des livrables de notre application en présentant les différents packages de notre projet.

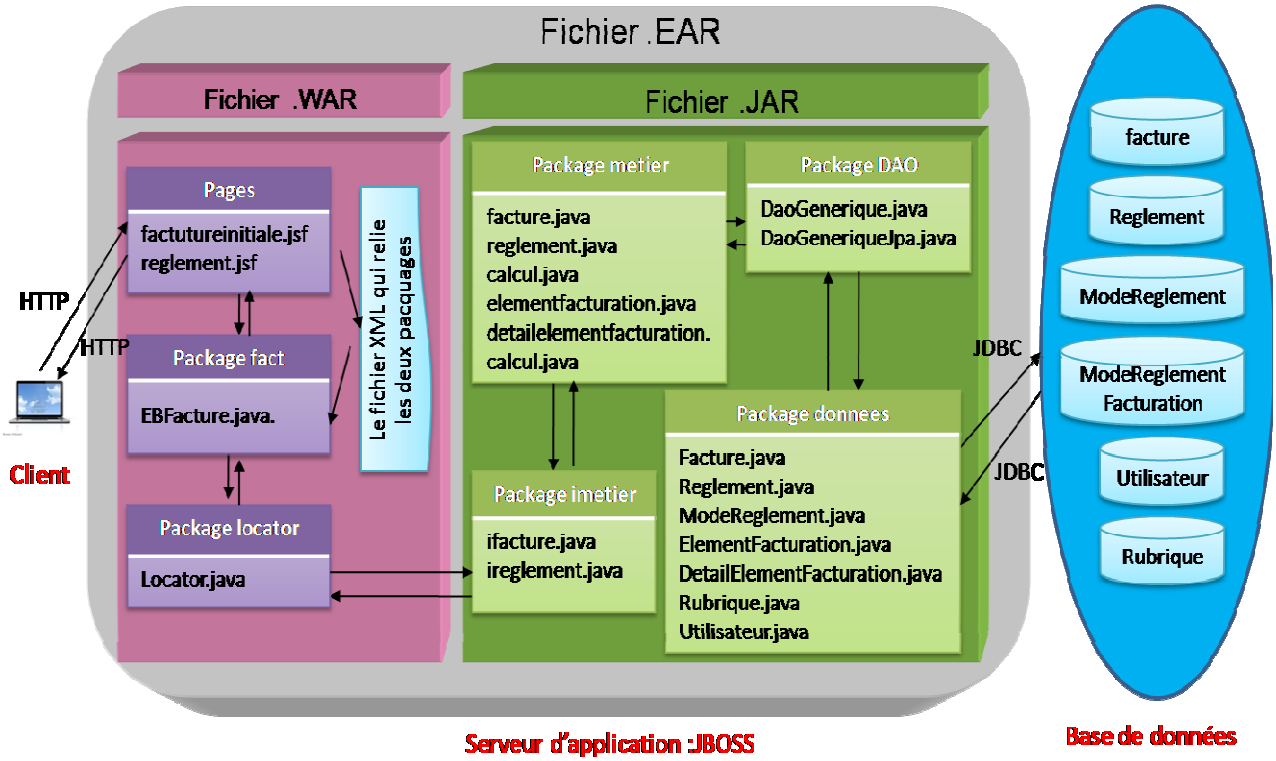


Figure II.3: Schéma applicatif détaillé de l'application.

Les différents packages de notre projet :

- Package fact, il contient le EBFacture.java.
- Package Locator.java, il contient la classe locator.java qui assure la communication entre le EBFacture.java et l'interface ifacture.java.
- Package imetier, il contient l'interface ifacture.java qui publie toutes les méthodes de la classe facture.java.
- Package metier, il contient les classes facture.java, reglement.java, elementfacturation.java et detailelementfaturation.java. Elles renferment toutes les méthodes de traitement.
- Package DAO, il contient deux classes : la classe DaoGeneriqueJpa.java qui implémente les quatre opérations de base CRUD et la classe DaoGenerique.java qui est une interface de DaoGeneriqueJpa.java.
- Package donnee, c'est une image de la base de données sous forme de classe.java : Facture.java, Reglement.java, ModeReglement.java, ElementFacturation.java, DetailElementFacturation.java, Utilisateur.java, Rubrique.java.
- Un dossier de nos pages JSF : factureinitiale.jsf, reglement.jsf, etc.

- La classe calcul est factorisée. Elle est appelée par les classes métier (facture.java, elementfactuartion.java et detailelementfacturation.java) pour calculer :
    - Le montant TVA (Taxe sur la Valeur Ajoutée), le montant THT (Total Hors Taxe), le montant TTC (Toutes Taxes Comprises) de la facture initiale et pro format.
    - Le montant TVA, le montant TTC des rubriques de la facture.

4. Principe de fonctionnement de l'application

Au chargement de la page factureinitiale.jsf, la classe Locator.java s'instancie, pour créer un lien de communication entre le ManagedBean EBfacture.java et l'interface métier ielementfacturation.java et detailelementfacturation.java. Ainsi toutes les méthodes des classes métier elementfacturation.java et detailelementfacturation.java publiées dans la ielementfacturation.java et idetailelementfacturation.java seront accessibles et visibles par EBfacture.java.

Grâce à l'interface DaoGenerique.java, les méthodes de la classe DaoGeneriqueJpa.java seront accessibles et visibles par les classes métier elementfacturation.java, detailelementfacturation.java et facture.java.

Remarque :

- Dans le schéma qui suit, on peut passer de l'état 3 à l'état 5 si le client est conventionné. C'est-à-dire qu'il peut ne pas régler une partie ou la totalité de sa facture.
- Si le client n'est pas conventionné on ne peut pas accéder à l'état 5 jusqu'à ce que le montant restant soit égal à zéro.

(Voir figure II.4 : Principe de fonctionnement de l'application).

- Même s'il n'est pas schématisé dans la figure II.4, le processus d'édition de la facture initiale est automatiquement activé dans notre application juste après le règlement de cette facture (état 6).

5. Diagramme de classes de l'application

La figure ci-dessous présente l'ensemble des classes de l'application réalisée, en spécifiant quelques méthodes et attributs utilisées.

(Voir figure II.6 : Diagramme de classes de l'application).

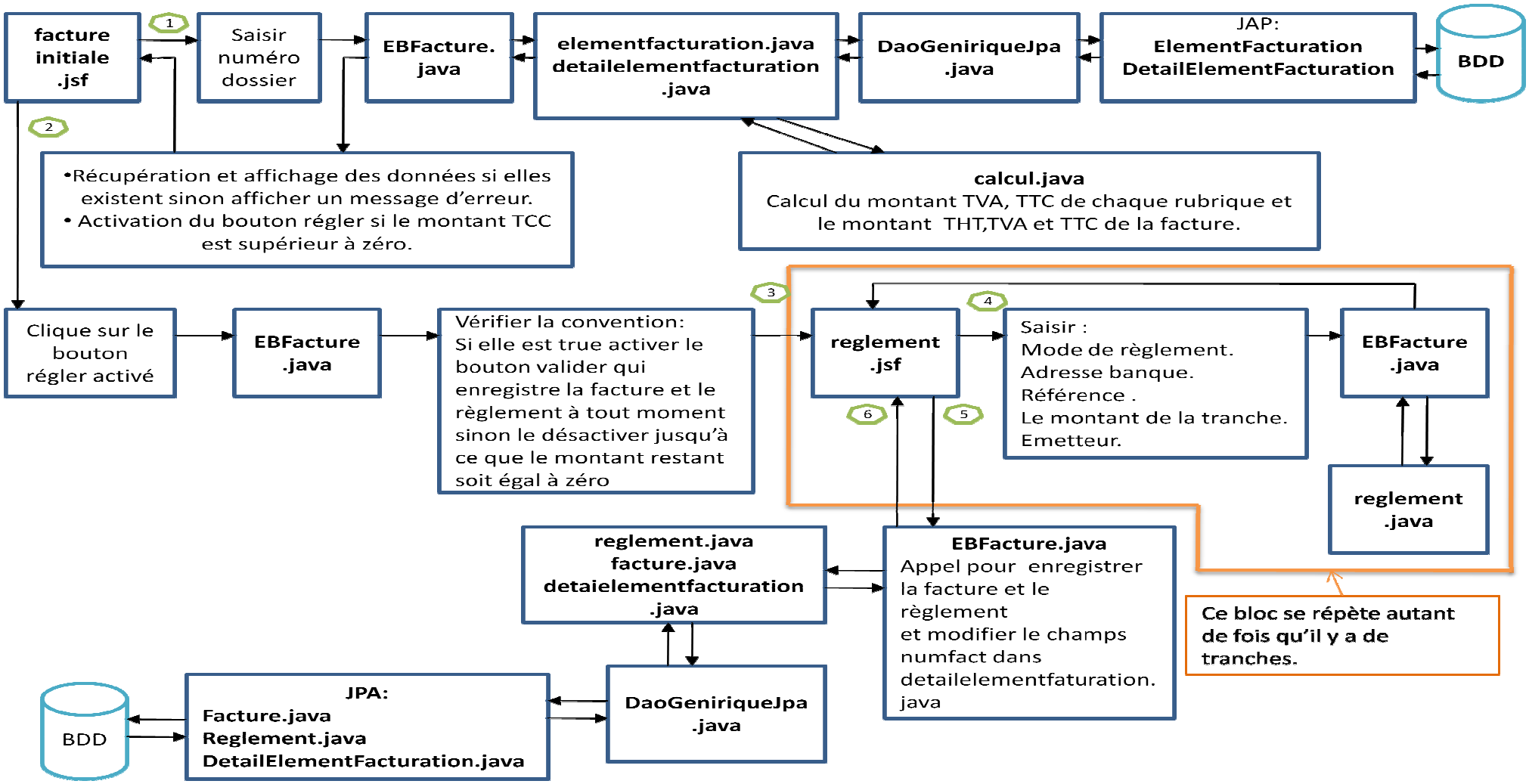


Figure II.4: Principe de fonctionnement de l'application.

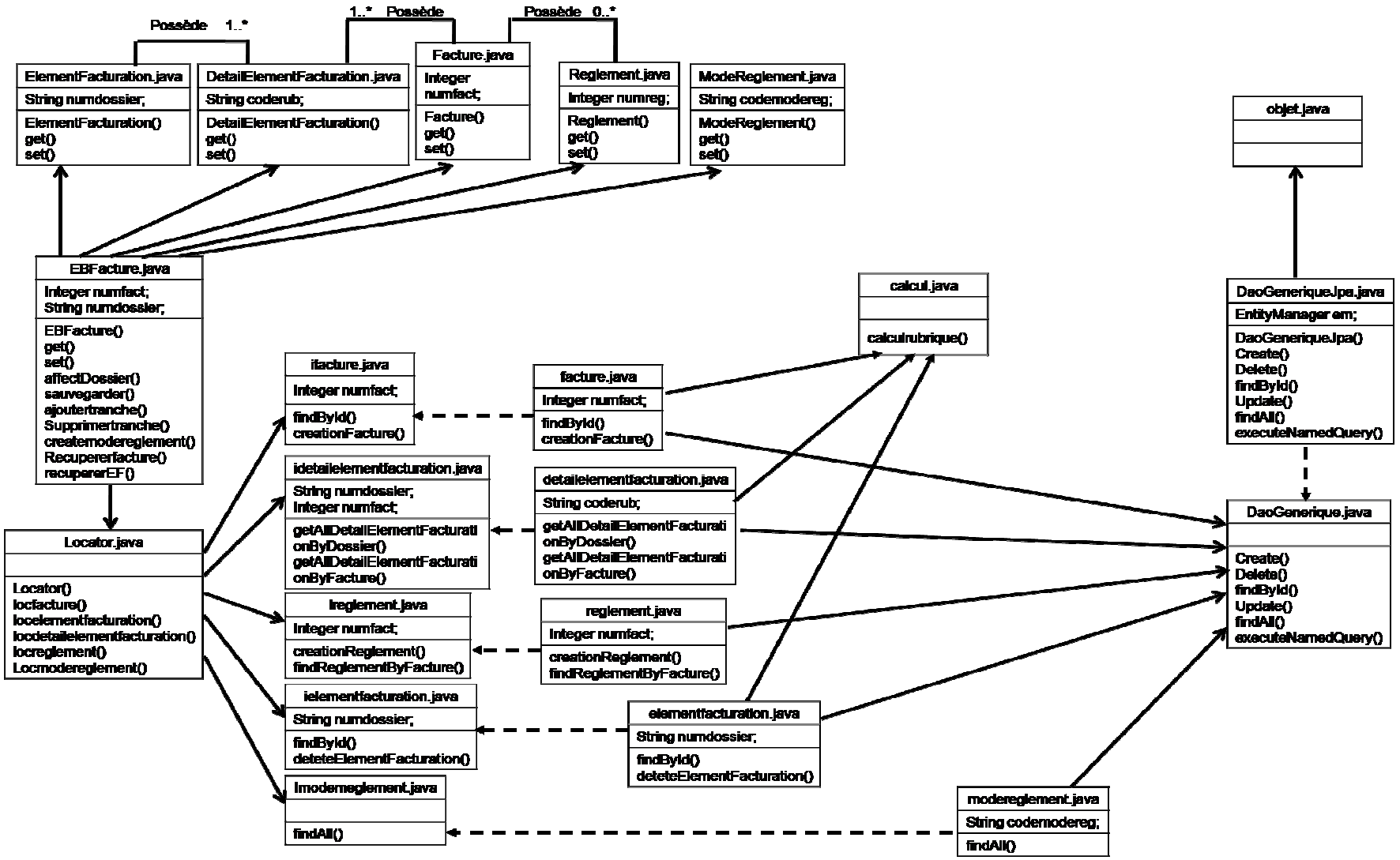


Figure II.5: Diagramme de classes de l'application.

6. Présentation de quelques interfaces

L'application que nous avons réalisé se présente sous une fenêtre principale qui permet à l'utilisateur de d'authentifier pour qu'il puisse accéder au menu principal puis aux fonctionnalités que comporte notre application.

6.1. Interface d'authentification

L'utilisateur doit saisir son nom d'utilisateur et son mot de passe. Si ces deux données sont correctes, le menu principal de notre application sera affiché sinon, un message d'erreur est affiché en rouge sur le formulaire d'authentification précisant à l'utilisateur l'erreur détectée.

L'interface d'authentification est représentée ci-dessous.

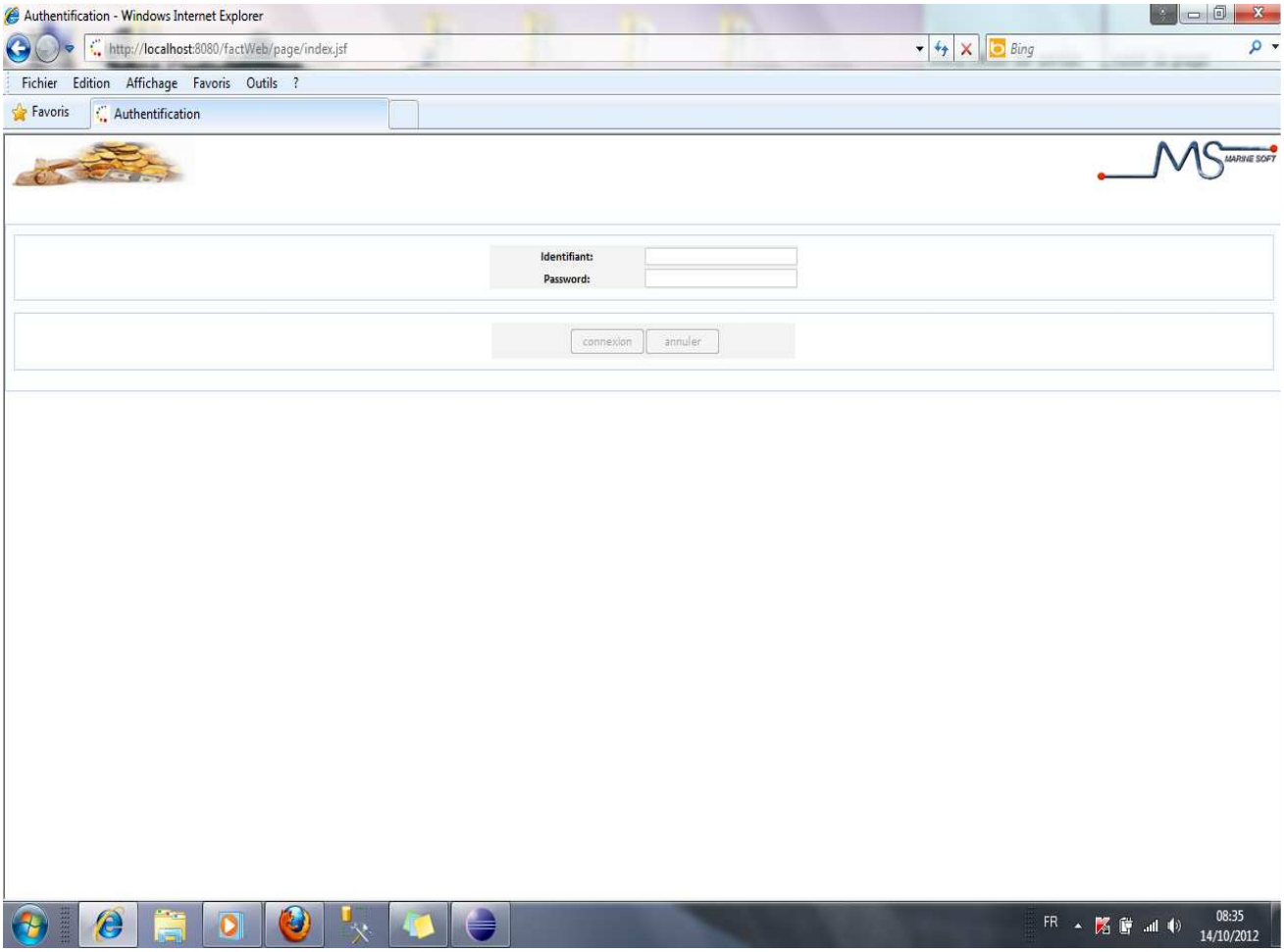


Figure II.6: Interface « authentification ».

6.2. Interface du menu principal

Une fois l'utilisateur identifié grâce à son nom d'utilisateur et son mot de passe, le menu apparait et donne accès à 4 rubriques en principe :

- Caisse qui est composée de :
    - Facture pro format.
    - Facture initiale.
    - Facture avoir.
    - Facture complémentaire.
- Suivi des factures à crédit : on a prévu une gestion, pour les clients n'ont pas encore réglé leurs factures, dans le cas des clients conventionnés.
- Réédition, on peut rééditer :
    - Les factures.
    - Etat de caisse entre deux dates.
    - L'historique d'une facture.
    - Un bon de reçu.
- Recherche multi critères : c'est un outil de recherche des factures (par un numéro de la facture, numéro de dossier ou date de facturation...etc.) ou des règlements (par le numéro du règlement ou date de règlement).

La figure suivante illustre cette page :

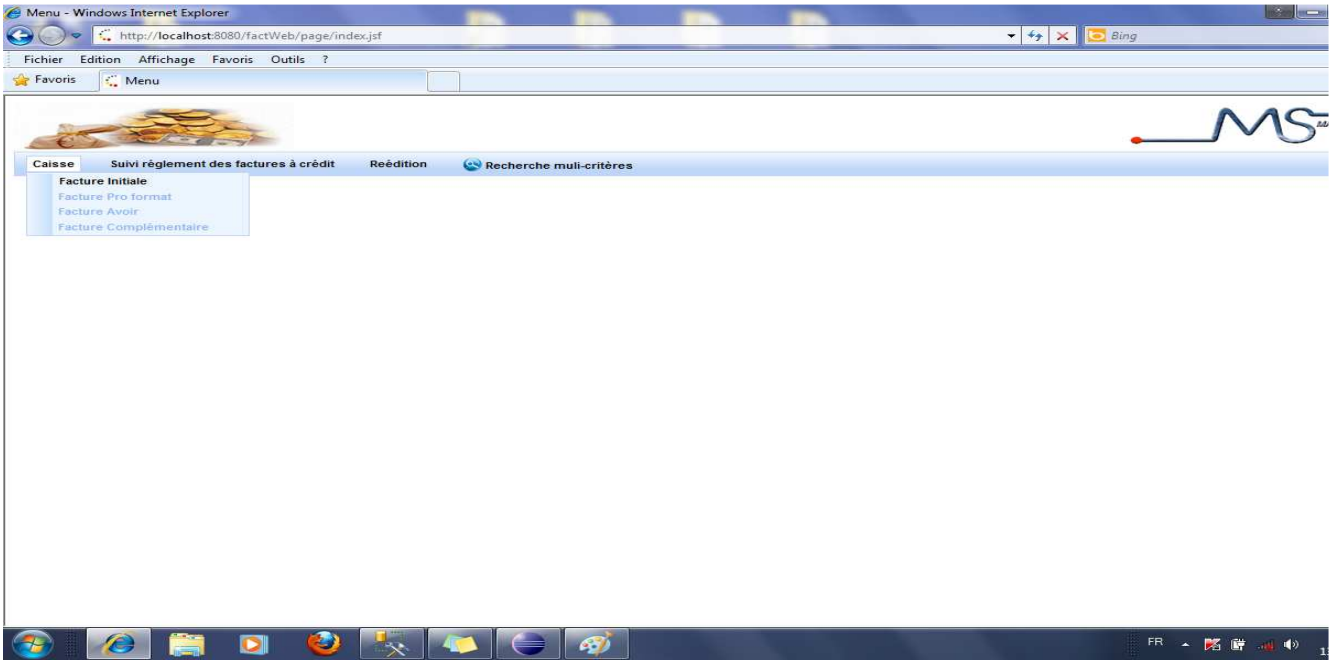


Figure II.7: Interface « menu ».

6.3. Interface de la facture initiale

Voici le formulaire de la facture initiale.

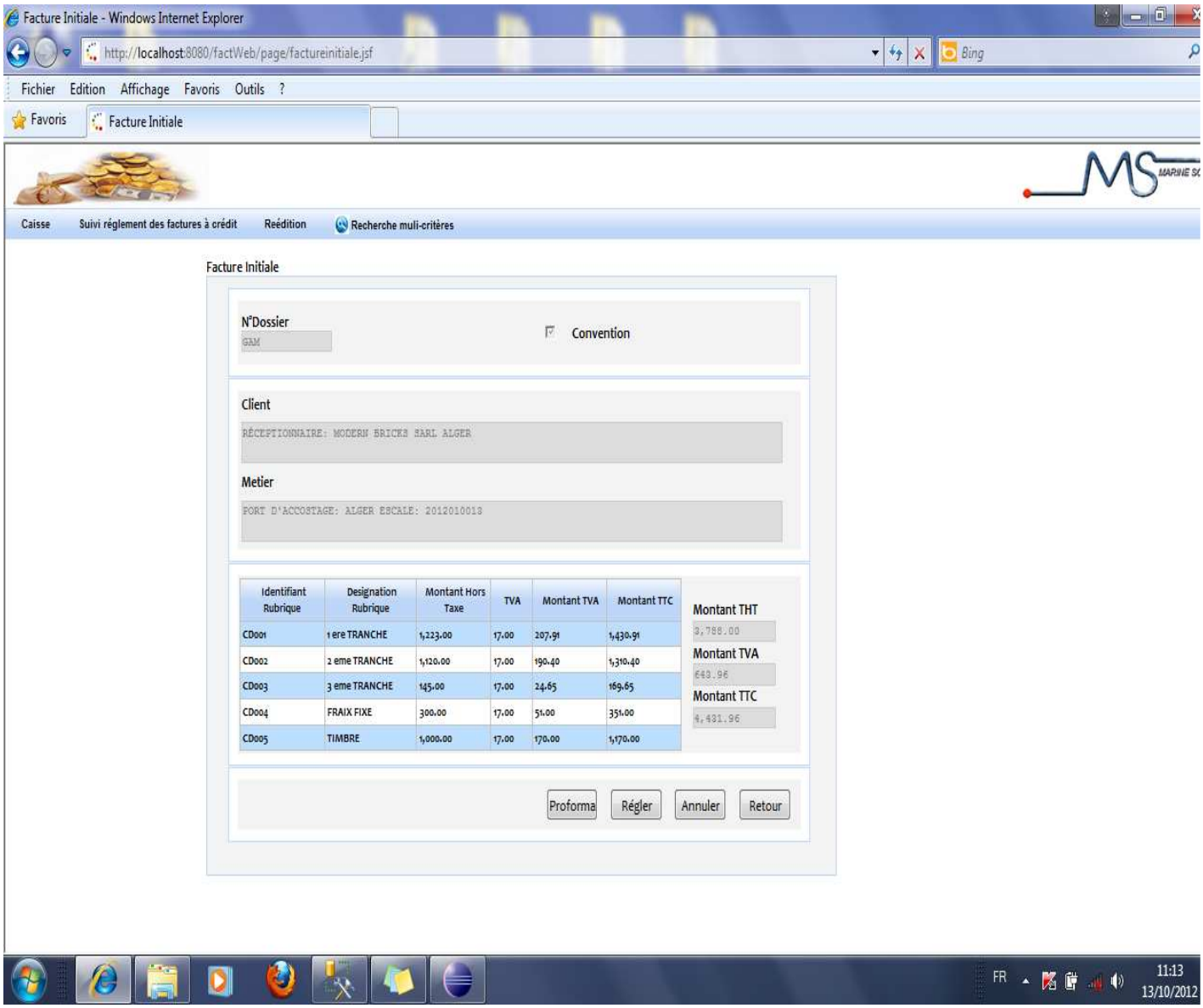


Figure II.8: Interface « facture initiale ».

L'utilisateur doit saisir le numéro de dossier, pour que le système lui affiche les informations depuis la base de données. Si celles-ci existent et que le montant TTC est supérieur à zéro le bouton régler s'active pour qu'il puisse accéder au règlement.

6.4. Interface du règlement

Voici le formulaire du règlement :

L'utilisateur doit saisir toutes les informations nécessaires pour qu'il puisse régler sa facture initiale en tranche. Sachant que le client conventionné a le choix de ne pas régler sa facture avant l'enregistrement. Contrairement au client non conventionné qui doit régler la totalité de sa facture pour qu'il puisse l'enregistrer dans la base de données.

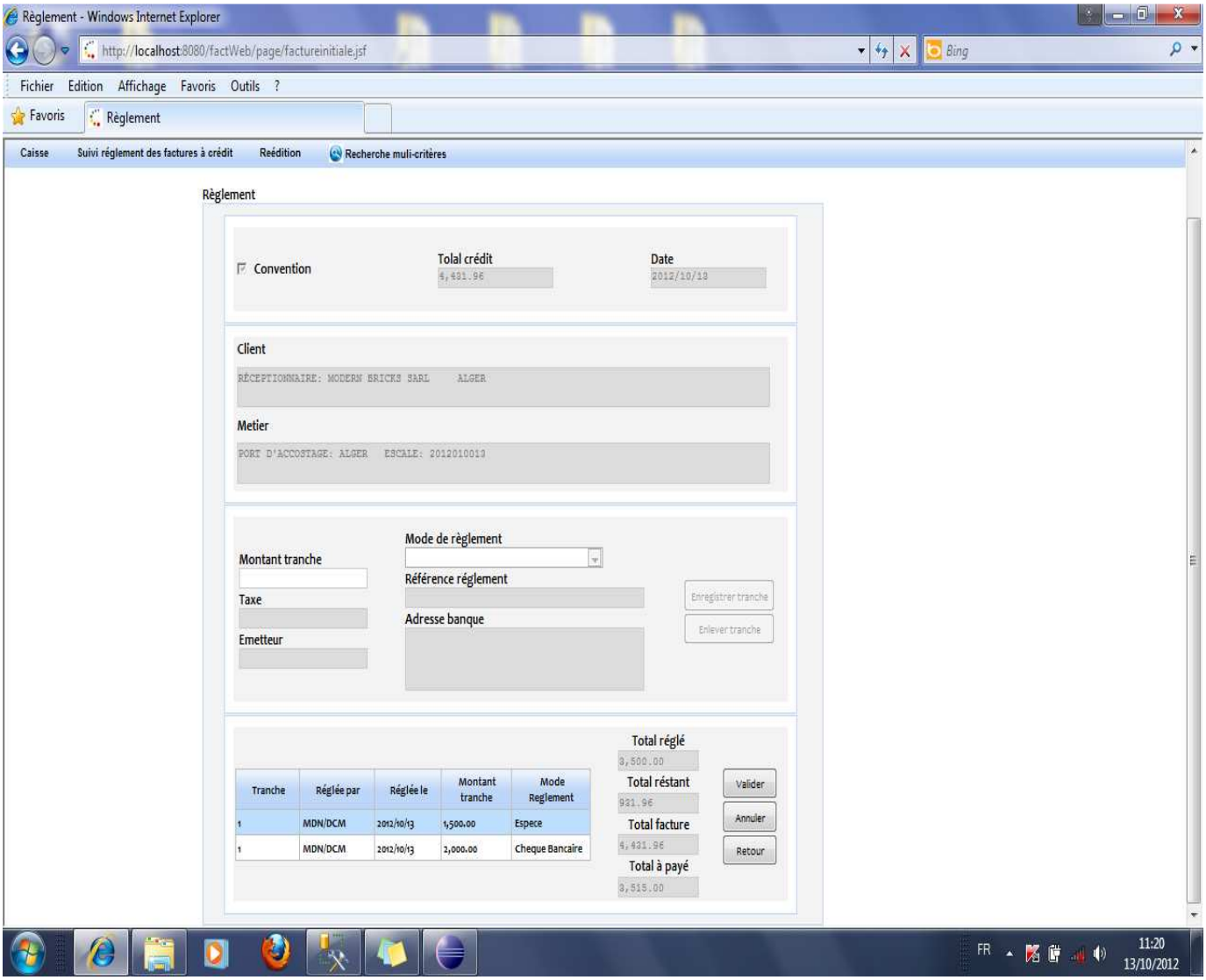


Figure II.9: Interface « règlement ».

Conclusion

Ce chapitre a pour objectif de dévoiler la complexité de la réalisation de la facture initiale. Il nous a permis de mettre en pratique les notions théoriques de Java EE. Autrement dit, nous avons pu séparer la couche présentation, couche métier et celles de la couche base de données.

Conclusion générale

La conception et la réalisation d'un système d'information standard pour la gestion de facturation, un module important du noyau de l'ERP, était la mission qui nous a été confiée.

Les étapes qui ont été suivies pour l'aboutissement de ce travail étaient :

- L'analyse et l'étude de l'existant, qui nous ont permis de prendre connaissance de la situation des logiciels produits par l'organisme d'accueil.
- L'identification des besoins qui nous a permis de concevoir notre application, la modéliser en utilisant le langage UML.
- La réalisation de l'application en se basant sur la plate-forme Java EE.

Ce travail nous a été bénéfique car il nous a permis d'acquérir de nouvelles connaissances et une certaine expérience au sein d'un nouveau milieu, d'appliquer nos connaissances théoriques acquises durant notre cursus universitaire, et aussi de :

- Elargir nos connaissances et découvrir une des architectures d'application distribuée à base de composants qui est Java EE.
- Se familiariser avec les outils de développement et de nouveaux Frameworks: Eclipse, Jboss, JSF…etc.
- S'adapter au travail de groupe.

Ce travail n'est pas une fin en soi. Il peut toutefois être complété. En effet, il ne couvre qu'une partie des fonctionnalités conçues. De ce fait, des perspectives d'amélioration suivantes sont ouvertes :

- L'implémentation des fonctionnalités autres que la facture initiale.
- L'implémentation d'une couche de communication avec les autres modules de l'ERP.
- L'implémentation d'une couche de communication avec un logiciel de comptabilité.

Ce projet informatique reste une modeste contribution dans le domaine des applications web.

Nous espérons qu'il constituera une source d'inspiration bénéfique pour les futurs étudiants qui aborderont ce genre de thème.

Annexe

- *Annexe I : Architecture Client/serveur*
- *Annexe II : La plate forme Java EE*
- *Annexe III : Design Pattern*

Annexe I

Architecture Client/serveur

- *Généralités sur le modèle client/serveur*
- *Client /serveur 2tiers*
- *Client /serveur 3tiers*
- *L'architecture N tiers*

Sommaire

La liste des figures

Introduction

Depuis son apparition, le client/serveur est devenu aujourd'hui l'architecture d'application la plus prisée. Avec le client/serveur, sont réunis des traitements locaux et distants dans une même application afin d'obtenir le meilleur des deux. Le client/serveur a révolutionné la manière dont étaient conçues jusqu'ici les applications et ce grâce au principe de la répartition des charges entre clients et serveurs ainsi qu'a la facilité d'utilisation offertes aux utilisateurs. En réalité le client/serveur a évolué en deux grandes étapes en passant du client/serveur de données au client/serveur de données et de procédures. Aujourd'hui la popularité du client/serveur ne cesse de croître et on assiste à un mouvement dit l'informatique client/serveur qui est extrêmement actif et qui remodèle l'utilisation de l'ordinateur.

L'objectif du présent chapitre est de rappeler le principe de fonctionnement du modèle client/serveur, de mettre en évidence ses avantages et ses limites. Nous verrons aussi qu'aujourd'hui les fonctions sont distribuées entre le client et le serveur et que ce partage de fonctionnalités conduit à une nouvelle orientation du modèle client/serveur qui sera dit orienté client ou orienté serveur (plus répandu).

Partant de ce principe de séparation de fonctionnalités, on est passée des applications monolithiques au client/serveur deux-tiers pour arriver aujourd'hui aux applications basées sur des architectures client/serveur multi-tiers. Enfin, nous verrons que l'architecture multi-tiers (dite aussi à base de composants) est la plus adéquate pour le développement des applications complexes qui ne peuvent plus être mises en place avec le modèle classique.

1. Généralités sur le modèle client/serveur

1.1. Présentation de l'architecture client/serveur

L'architecture client/serveur s'articule en général autour d'un réseau. Le serveur assure la gestion des données partagées entre les utilisateurs. Le client gère l'interface de la station de travail d'un utilisateur. Les deux communiquent par des protocoles et les programmes applicatifs sont sur le client et/ou le serveur.

Une autre définition plus large peut être proposée : « Modèle d'architecture applicative ou les programmes répartis entre processus clients et serveurs communiquent par des requêtes et des réponses ». On définit trois acteurs principaux pour l'architecture client/serveur (voir figure I.1 Client/Médiateur/Serveur) :

- ***Le client** : processus demandant l'exécution d'une opération à un autre processus serveur par l'envoi d'un message contenant le descriptif de l'opération à exécuter et attendant une réponse à cette opération par un message en retour.*
- ***Le serveur** : processus accomplissant une opération sur demande d'un client et transmettant la réponse à ce client.*
- ***Le middleware** : ensemble des services logiciels construits au-dessus d'un protocole de transport afin de permettre l'échange de requêtes et de réponses associées entre le client et le serveur de manière transparente.*

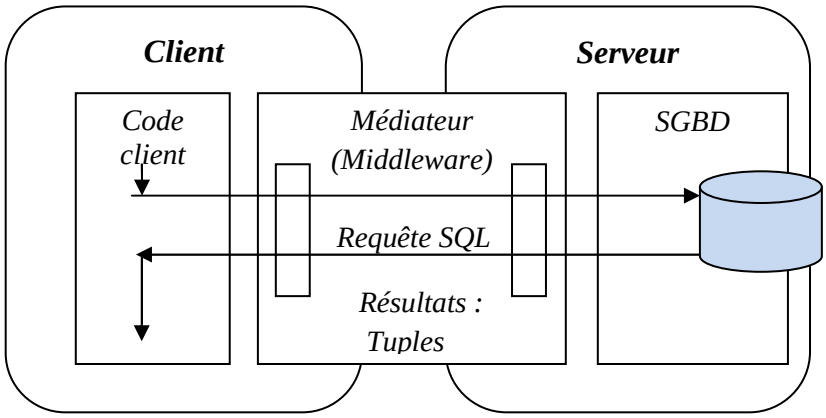


Figure I.1.*Client/médiateur/serveur*

1.2. Propriétés de l'architecture client/serveur

Le terme client/serveur est devenu le maître mot en informatique. Cette technologie se distingue des autres systèmes informatiques répartis par un ensemble de propriétés qui la caractérise :

- ***Le service :*** *c'est le travail fourni par le serveur suite à la requête du client. Le client est donc consommateur et le serveur fournisseur de services.*
- ***Les ressources partagées*** *: le serveur est capable de servir de nombreux clients simultanément et réguler leur accès aux ressources.*
- ***Les protocoles asymétriques*** *: c'est toujours le client qui déclenche la demande de service. Le serveur attend passivement les requêtes des clients.*
- ***La transparence*** *: le serveur peut résider ou non sur la même machine que le client, sans différence pour ce dernier.*
- ***La compatibilité*** *: l'application Client ou Serveur est indépendante du matériel.*
- ***Le faible couplage*** *: le client et le serveur communiquent uniquement par envoi de messages.*
- ***L'encapsulation des services*** *: Le serveur choisit la manière dont il réalise le service demandé, le client se borne à définir ce qu'il désire obtenir. Ainsi l'implémentation du serveur peut être changée sans préoccuper le client.*
- ***L'adaptabilité*** *: des machines peuvent être ajoutées ou retirées du réseau, les performances des machines peuvent aussi évoluer.*

1.3. L'architecture mainframe et l'évolution du client/serveur

Avant que n'apparaisse le mode client-serveur (avant les années 80), les réseaux informatiques étaient configurés autour d'un ordinateur central appelé "Mainframe" auquel étaient connecté des terminaux passifs (écran adjoint d'un clavier sans unité centrale). Tous les utilisateurs sont alors connectés sur la même unité centrale et partagent donc les mêmes traitements informatiques et données. Le "Mainframe" n'affiche que du texte à l'écran sans graphisme (pas de bouton, pas de fenêtre,…). Il est spécialisé dans la gestion d'information de masse auquel il peut appliquer des instructions simples (addition, soustraction…).

Les années 80 ont connu le développement du transactionnel et des bases de données ainsi que l'ouverture des systèmes qui étaient jusque là propriétaires.

A partir des années 90, le besoin de partage des données devient essentiel aussi bien pour l'accès transactionnel caractérisé par des mises à jour rapides en temps réel, que pour l'accès décisionnel marqué par le besoin de requêtes sur de gros volumes de données.

1.4. Composition d'une architecture client/serveur

Trois modules composent une architecture client/serveur. Ces modules sont repartis sur le client et sur le serveur :

- *Un module interface utilisateur ou présentation.*
- *Un module de traitement ou de logique applicative.*
- *Un module de gestion de données.*

1.5. Les différents modèles de client-serveur

En fait, les différences sont essentiellement liées aux services qui sont assurés par le serveur. On distingue couramment ***:***

1.5.1. Le client-serveur de donnée

Dans ce cas, le serveur assure des tâches de gestion, de stockage et de traitement de données. C'est le cas le plus connu du client-serveur est qui est utilisé par tous les grands SGBD.

1.5.2. Client-serveur de présentation

Dans ce cas la présentation des pages affichées par le client est intégralement prise en charge par le serveur. Cette organisation présente l'inconvénient de générer un fort trafic réseau.

1.5.3. Le client-serveur de traitement

Dans ce cas, le serveur effectue des traitements à la demande du client. Il peut s'agir des traitements particuliers sur des données, de vérification de formulaires de saisie, des traitements d'alarmes …

1.5.4. Une synthèse des différents cas

Cette synthèse s'illustre par un schéma du Gartner Group qui représente les différents modèles ainsi que la répartition des tâches entre serveur et client.

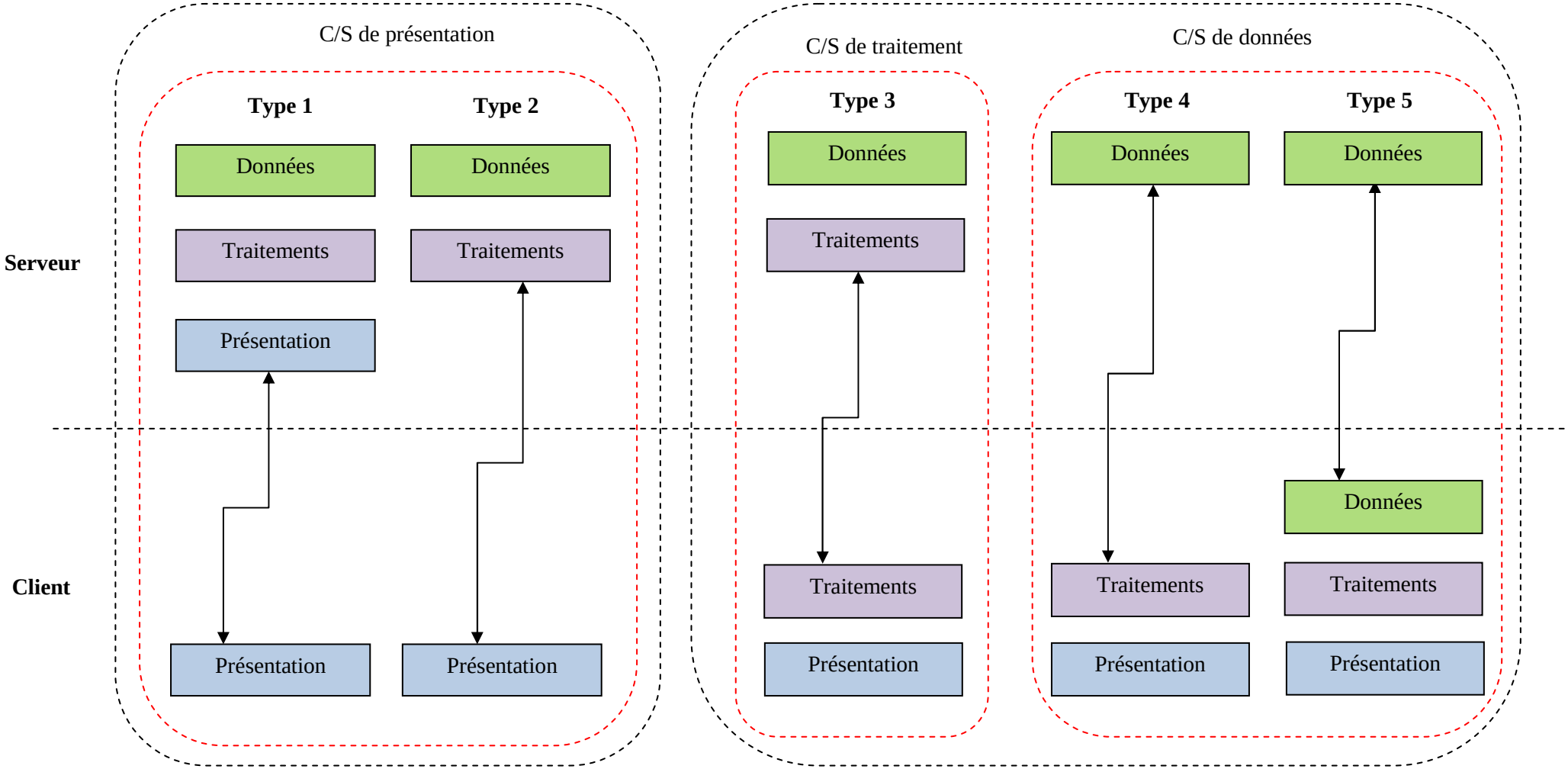


Figure I.2. : *Les différents types de client/serveur selon la classification de Garther Group*

1.6. Type de client

- ***Client léger :***

Le terme client léger (parfois client pauvre, thin client) désigne une application accessible via une interface web (en HTML) consultable à l'aide d'un navigateur web, où la totalité de la logique métier est traitée du coté serveur.

- ***Client lourd :***

Le terme client lourd (fat client ou heavy client), par opposition au client léger, désigne un client graphique exécuté sur le système d'exploitation de l'utilisateur.

Un client lourd possède généralement des capacités de traitement évoluées et peut posséder une interface graphique sophistiquée

- ***Client riche:***

Un client riche est un compromis entre le client léger et le client lourd. L'objectif du client riche est donc de proposer une interface graphique, décrite avec une grammaire de description basée sur la syntaxe XML, permettant d'obtenir des fonctionnalités similaires à celles d'un client lourd (glisser-déposer, multifenêtrage, etc.).

Les clients riches permettent ainsi de gérer l'essentiel des traitements du côté serveur. Les données sont ensuite transmises dans un format d'échange standard utilisant la syntaxe XML, puis interprétées par le client riche.

1.7. Types de serveurs

Il existe plusieurs serveurs, les plus utilisés sont :

- ***Serveurs de fichiers :***

Le serveur s'occupe de la gestion des fichiers. Le client demande l'accès (écriture ou lecture) à des fichiers en émettant des requêtes sur un réseau en direction du serveur. Les serveurs de fichiers sont utiles pour partager des fichiers sur un réseau et ils sont indispensables pour créer des banques de documents, d'images…

- ***Serveur de base de données :***

Dans le cas du serveur de base de données, le client émet des requêtes SQL sous forme de messages en direction du serveur. Les données ainsi que le code qui traite les requêtes résident sur la même machine serveur. Le stockage et le traitement de ces données sont réalisés par le serveur.

Exemple : Oracle, Microsoft Server SQL.

- ***Serveur de transaction :***

Une transaction correspond à une procédure SQL, c'est-à-dire un groupe d'instruction SQL. Dans ce modèle, les clients invoquent ces procédures distantes résidentes sur le serveur qui exécutent un ensemble d'instructions SQL.

Il y a un seul échange requête/réponse pour la transaction (une réponse pour un bloc de requêtes SQL). Ces serveurs sont essentiellement utilisés pour l'informatique de production (ou opérationnelle) pour laquelle la rapidité des temps de réponse est importante.

2. Client /serveur 2tiers

2.1. Présentation

Dans l'architecture 2-tiers, la gestion des données est prise en charge par un SGBD centralisé, s'exécutant le plus souvent sur un serveur dédié alors que la logique applicative est enfouie dans l'interface graphique chez le client. Le dialogue entre le client et le serveur se résume à l'envoi des requêtes et au retour des données correspondant aux réponses. (Voir ***Figure AI.4.*** *Architectures client/serveur 2-tiers).*

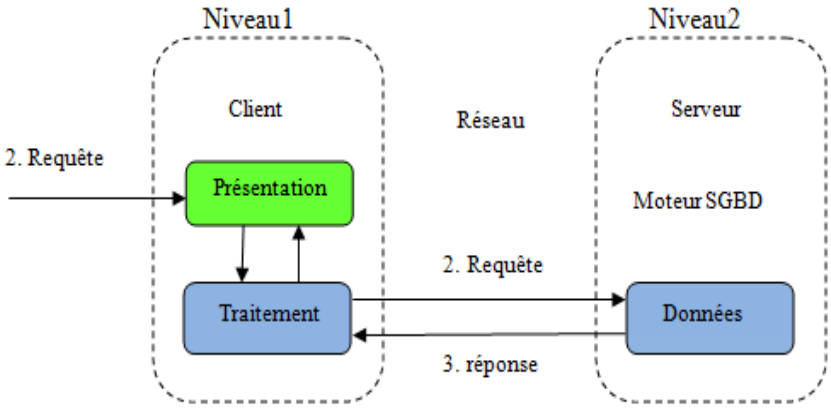


Figure I.3. *Architectures client/serveur 2-tiers.*

2.2. Limites du client/serveur 2-tiers

La technique simple du client/serveur 2-tiers est excellente pour créer, rapidement, des applications telles que les applications d'aide à la décision. Elle est considérée comme une bonne solution informatique lorsque le nombre d'utilisateurs ne dépasse pas la centaine. Ce modèle eut, en effet, un grand succès pour les petites applications (interface utilisateur riche, appropriation des applications par l'utilisateur, interopérabilité). Ce n'est qu'avec l'apparition du besoin de développement d'applications à missions critiques (à grande échelle) que le client/serveur 2-tiers montra ses limites et devint inadapté pour celles-ci. Le manque de modularité lorsque toute la logique applicative se trouve sur le poste client ne facilite pas la maintenance et la réutilisation. Une modification de l'application ou de la structure de la base de données nécessite forcément un redéploiement sur les postes clients. En effet, l'expérience a démontré qu'il était coûteux de vouloir faire porter l'ensemble des traitements applicatifs par le poste client qui devient de ce fait un client lourd. Les limites du client/serveur 2-tiers peuvent se résumer essentiellement à :

- *La charge, difficile à réduire, du poste client qui supporte la grande majorité des traitements applicatifs.*
- *Le poste client est fortement sollicité, il devient de plus en plus complexe et doit être mis à jour régulièrement pour répondre aux besoins des utilisateurs.*
- *La conversation entre le client et le serveur est assez bruyante et s'adapte mal à des bandes passantes étroites.*

Les applications d'aujourd'hui sont capables de servir des milliers de clients. Avec l'arrivée du commerce électronique, elles peuvent se déployer sur plusieurs entreprises et avec l'Internet, les serveurs

peuvent recevoir des requêtes émanant de millions de navigateurs répartis dans le monde entier. Le stade du client/serveur 2-tiers est, pour ce type d'application, bel et bien dépassé car il a débordé de son cadre originel. La transition vers un autre modèle d'architecture du client/serveur 3-tiers et même multi-tiers s'impose pour aborder ce monde complexe dans lequel les applications sont découpées en centaines de composants répartis sur de nombreux serveurs. Le modèle à 3 niveaux permet de dépasser ces limites et d'apporter une meilleure réactivité de l'entreprise en cas de changements.

Malgré tout, l'architecture deux tiers présente de nombreux avantages qui lui permettent de présenter un bilan globalement positif :

- *Elle permet l'utilisation d'une interface utilisateur riche ;*
- *Elle a permis l'appropriation des applications par l'utilisateur ;*
- *Elle a introduit la notion d'interopérabilité.*

Pour résoudre les limitations du client-serveur deux tiers tout en conservant ses avantages, on a cherché une architecture plus évoluée, facilitant les forts déploiements à moindre coût. La réponse est apportée par les architectures distribuées.

3. Client /serveur 3tiers

3.1. Présentation

Dans l'architecture à 3 niveaux, un niveau médian est ajouté entre les deux niveaux précédents, permettant de séparer les traitements de l'interface graphique et du serveur de base de données. Trois types de plates-formes doivent donc supporter ces composants, d'où l'architecture 3-tiers du client/serveur.

Autrement dit, avec le client/serveur 3-tiers, le client fournit l'interface graphique et interagit avec le serveur par des appels de services distants. La logique applicative se trouve dans le niveau médian. Les traitements deviennent des entités de première importance, gérées et déployées indépendamment de l'interface utilisateur et de la base de données. La logique applicative qui a maintenant son propre niveau séparé, peut même s'exécuter sur un ou plusieurs serveurs (Voir ***Figure : I.4.*** *Architecture client/serveur à 3-tiers).*

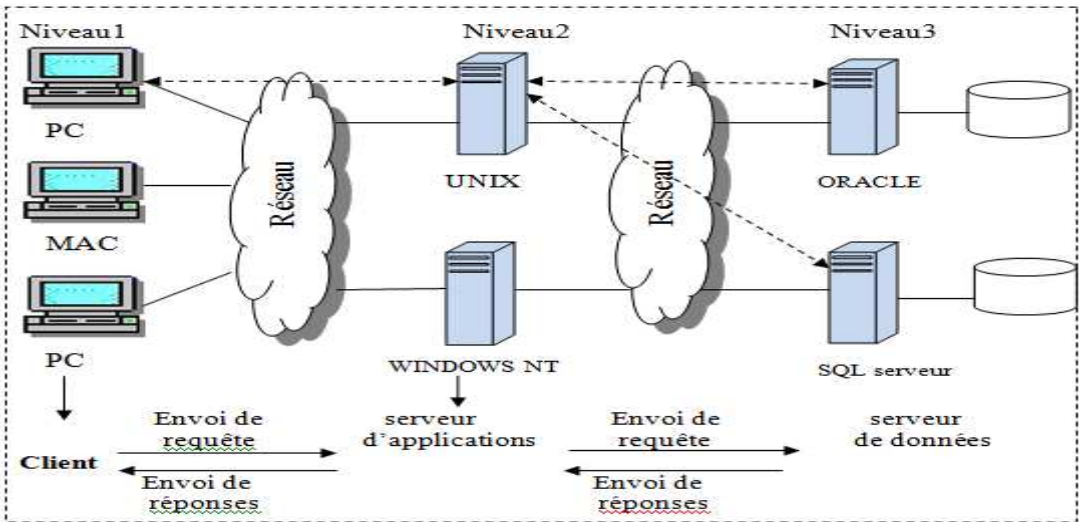


Figure I.4. *Architectures client/serveur à 3-tiers*

3.2. Avantages du client/serveur 3-tiers

C'est grâce au modèle à 3-tiers que le client/serveur se développe et s'étend, car ce modèle répond aux besoins vastes des applications client/serveur de l'Internet et des intranets. Cette architecture supporte des milliers d'utilisateurs accédant à plusieurs serveurs répartis géographiquement. La plus grande partie du code étant sur les serveurs, les applications 3-tiers deviennent plus faciles à gérer et à déployer sur le réseau. En outre les applications à 3 niveaux réduisent les échanges sur le réseau en créant des niveaux de service abstraits. Au lieu d'agir directement avec la base de données, le client appelle la logique applicative sur le serveur, laquelle accède alors à la base de données conformément à la demande du client. Le modèle 3-tiers remplace de nombreuses requêtes par un petit nombre d'appels au serveur. Il permet un accès spécifique au serveur et le client n'accède pas directement à la structure de la base de données. Il en résulte que ses performances et son niveau de sécurité sont plus élevés que ceux du modèle 2-tiers. En théorie, les systèmes client/serveur à trois niveaux sont mieux dimensionnables, plus robustes et plus souples. De plus, ils peuvent intégrer des données venant de sources multiples.

4. L'architecture N tiers

4.1. Présentation

Dans l'architecture 3-tiers, pour répondre à une demande de service émise par un client, un serveur peut utiliser les services d'un ou plusieurs autres serveurs afin de fournir son propre service (d'où le multi-tiers) (voir figure Client/serveur à N-tiers).

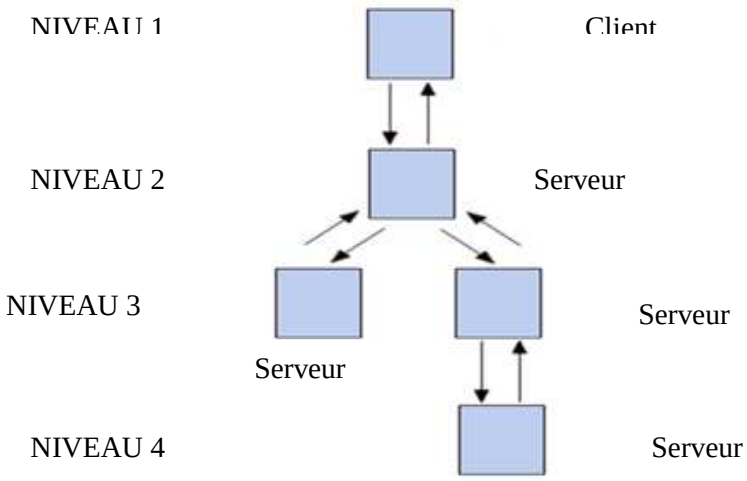


Figure I.5. *Architecture client /serveur à N niveaux*

Le niveau médian de la plupart des applications 3-tiers n'est pas implémenté sous forme d'un programme monolithique, mais comme un ensemble de composants utilisés lors de diverses transactions initiées par les clients. Chaque composant automatise une fonction relativement limitée. Les clients combinent souvent plusieurs composants du même niveau médian en une transaction opérationnelle unique. Un composant peut appeler d'autres composants pour implémenter une requête. Certains composants peuvent en outre se comporter comme

des passerelles encapsulant des applications. La plupart du temps, les 3-tiers sont donc en réalité N-tiers (voir figure Client/serveur N-tiers à base de composants).

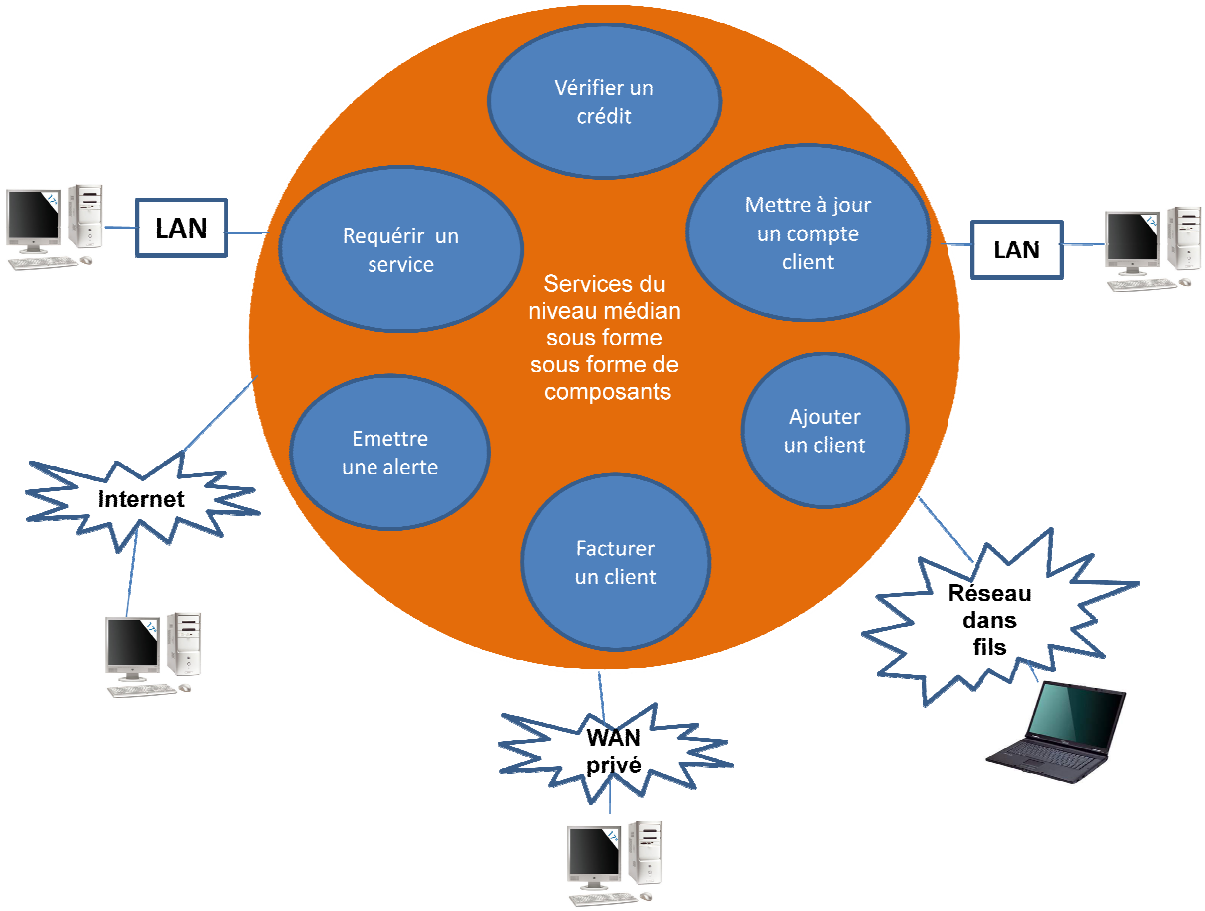


Figure I.6 *Client/serveur N-tiers à base de composants*

4.2. Les Avantages

En construisant le tiers médian à base de composants, plusieurs avantages sont offerts par rapport aux applications monolithiques :

- ***Développement de grandes applications par étapes :***

Les architectures basées sur les composants permettent de développer tant de grandes applications à missions critiques que de petits projets.

- ***Réutilisation des composants :***

Les composants sont ici réutilisables sous forme de 'boites noires' binaires qui peuvent être recombinées de diverses façons en fonction des applications.

- ***Accès fiable des clients aux données :***

Les clients émettent des requêtes vers des composants pour qu'une façon soit exécutée. Les composants serveur encapsulent le détail de la logique applicative, augmentant ainsi le niveau d'abstraction.

4.3. Le Middleware

Comme on vient de le voir, le temps des applications monolithiques est bien révolu. Une application peut désormais être constituée de centaines de morceaux qui auront été écrits indépendamment les uns des autres. Mais pour que le niveau médian (qui renferme les composants) agisse comme s'il était un système (ou une application) unique, il a besoin de middleware qui permettra d'unifier, pour les applications, l'accès et la manipulation de l'ensemble des services réseau.

4.3.1 Définition et objectif

Dans le cadre de l'informatique répartie, le middleware désigne toutes les couches logicielles qui permettent aux applications de communiquer à distance. Il doit permettre à un programme de s'exécuter sur plusieurs machines distantes. Il représente une intersection entre plusieurs domaines de l'informatique : Systèmes d'exploitation, réseaux, langages de programmation. Le middleware est un outil pour le développeur, enfoui dans les applications et qui n'est pas visible à l'utilisateur final. Il est considéré comme un complément de services réseau permettant la réalisation du dialogue client/serveur. La transparence aux réseaux, aux serveurs et aux langages est l'objectif principal du middleware :

- ***Transparence aux réseaux** : Tous les types de réseaux doivent être support´es, qu'ils soient locaux ou nationaux, voire internationaux. Le middleware cachera l'hétérogénéité des protocoles de transport utilisés en offrant une interface standard de dialogue à l'application.*

- ***Transparence aux serveurs** : Les SGBD mis en œuvre peuvent être très divers, offrant des moyens de connexion et des langages de requêtes souvent différents. Le médiateur doit cacher la diversité et uniformiser ces langages en s'appuyant le plus possible sur les standards.*

- ***Transparence aux langages :** Le médiateur doit permettre l'intégration de fonctions de connexion aux serveurs, d'´emissions de requêtes et de réception de réponse dans tous les langages de développement utilisés coté client.*

Le middleware est ainsi susceptible de rendre plusieurs services :

- ***Communication :** Permet la transmission de messages entre client et 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 de l'utilisateur.*
- ***Adressage :** Permet d'identifier la machine serveur sur laquelle est localisé le service demandé afin d'en déduire le chemin d'accès.*
- ***Conversion :** Service utilisé pour la communication entre les machines mettant en œuvre des formats de données différents (hétérogénéités des plates-formes).*
- ***Sécurité:** Permet de garantir la confidentialité et la sécurité des données grâce à des mécanismes d'authentification et de cryptage des informations.*
- ***Administration :** Permet la configuration et l'exploitation de l'ensemble des composants (système et application) des différentes plates-formes, généralement à partir d'un point central.*

Conclusion

Suite à ce que l'on vient de voir, on peut dire que le client/serveur est une évolution des applications monolithiques vers des applications réparties entre les processus client et serveur. Cette répartition des charges applicatives a fait émerger les notions d'orientation client et orientation serveur. Lorsque le client/serveur est orient´e client, la charge applicative tourne en majorité sur le client. Par contre, si le client/serveur est orient´e serveur, la charge applicative est répartie entre le client et le serveur : les clients exécutent l'interface graphique utilisateur, le serveur d'application exécute la logique applicative et gère la base de données. Le principe de répartition des charges dans le client/serveur améliore l'ouverture du système d'information d'entreprise et accélère le déploiement des architectures réparties et hétérogènes.

Aujourd'hui, l'architecture client/serveur a évolué vers l'architecture répartie, o`u le client peut s'adresser à différents serveurs et un serveur peut `a son tour devenir client d'un autre serveur. Le client/serveur est devenu une forme dominante de l'informatique et le modèle multi-tiers est devenu la forme dominante du client/serveur.

Le partitionnement en niveaux est le point clé de la conception d'une solution informatique client/serveur. C'est lui qui fera l'échec ou le succès des applications complexes. La connaissance des diverses architectures client/serveur permet d'éviter des erreurs d'architecture irrémédiables.

Même si la popularité de l'architecture multi-tiers s'accroît rapidement, le modèle 2-tiers est loin d'être obsolète. Il existe un grand nombre d'applications idéales pour les architectures 2-tiers. En effet, pour les plus petits projets, les applications 2-tiers sont plus faciles à développer que celles 3-tiers (ou n-tiers).

En général, dans le monde des applications client/serveur l'architecture multi-tiers occupe la place du favori. Elle permet de commencer à petite échelle, en dimensions et en fonctions, puis de faire grossir l'application jusqu'à ce qu'elle atteigne les proportions désirées. Elle permet également de créer un catalogue de composants, développés ou achetés, que l'on assemblera rapidement pour produire de nouvelles applications.

Annexe II

La plate forme Java EE

- *Pourquoi utiliser une plate-forme ?*
- *Définition de Java EE*
- *Pourquoi utiliser Java EE ?*
- *L'architecture Java EE en couche*
- *Le serveur d'application Java EE*

Sommaire

La liste des figures

Introduction

De nombreuses possibilités existent pour réaliser des applications Internet depuis plusieurs années. Des langages ont été crées, des architectures et des environnements de travail ont été conçus pour répondre aux besoins et faciliter la tâche des développeurs. Sun (le concepteur de Java) a donc mis en place un ensemble de technologies pour réaliser des applications Web. Ces technologies sont regroupées sous le nom J2EE (Java 2 Entreprise Edition), désormais Java EE.

1. Pourquoi utiliser une plate-forme ?

Une plateforme est une base générique qui fournit un ensemble de fonctionnalités utiles pour une majorité d'applications. Il peut exister plusieurs types de plates-formes, de la plus générique à la plus spécifique (optimisée pour un type de métier précis par exemple).

L'avantage principal d'utiliser une plate-forme est que l'équipe de développement n'a pas à s'acquitter de développer certaines tâches (connexion à la base de données par exemple, gestion d'objets …). Ce sont des tâches que l'on retrouve très souvent dans un grand nombre de projets et qui n'ont pas d'intérêts à être recoder à chaque fois (perte de temps et d'argent). De plus, mieux vaut travailler sur une plate-forme qui présente une forte stabilité (ça évite de debugger inutilement !).

Un autre avantage est la facilité de prise en main des APIs de cette plate-forme. En effet, celle-ci cache très souvent la complexité d'accès à telle ou telle ressource et permet donc un gain de temps énorme pour le développeur qui a donc plus de temps pour se préoccuper du fonctionnement réel de son application.

2. Définition de Java EE

La plate-forme Java EE s'appuie entièrement sur le langage Java. Java EE est donc une norme, qui permet à des développeurs de réaliser leur propre application qui implémente en totalité ou partiellement les spécifications de SUN. En simplifiant, il est possible de représenter Java EE comme un ensemble de spécifications d'APIs (ensemble de librairies et de services), une architecture et une méthode de packaging.

Remarque:

Depuis la version 5 de J2EE, le chiffre 2 a disparu pour faciliter la compréhension de la version et ne pas mélanger le chiffre 2 avec le numéro de version.

3. Pourquoi utiliser Java EE ?

Il existe actuellement beaucoup d'autres plates-formes de développement qui sont basées sur d'autres langages (C#, PHP5, .NET ...). Les principaux avantages d'utiliser Java EE (est donc Java) sont la portabilité, l'indépendance, la sécurité et la multitude de librairies proposées. Le développement d'applications d'entreprise nécessite la mise en œuvre d'une infrastructure importante. Beaucoup de fonctionnalités sont utilisées et développées, le but étant de produire des applications sures, robustes et faciles à maintenir. Certains services

sont d'ailleurs récurrents comme : l'accès aux bases données, l'envoi de mails, les transactions, la gestion de fichiers, la gestion d'images, le téléchargement, le chargement et la supervision du système...

C'est pour cela que l'architecture Java EE est intéressante car tous les éléments fondamentaux sont déjà en place. Pas besoin de concevoir une architecture, des librairies et des outils spécialement adaptés. Cela nécessite un temps et un investissement considérables.

Enfin, la plate-forme Java EE est basée sur des spécifications, ce qui signifie que les projets sont portables sur n'importe quel serveur d'application conforme (Tomcat, JBoss, WebSphere et Weblogic...) à ces spécifications. Cette implémentation est gratuite et permet de bénéficier de la totalité de l'API sans investissement. La plate-forme Java EE est la plus riche des plates-formes Java et offre un environnement standard de développement et d'exécution d'applications d'entreprise multi-tiers.

Le fait que Java EE soit standardisé, il a contribué à son adoption par de très nombreux éditeurs de logiciels/outils informatique. Ces éditeurs associés à Sun Microsystems font partis du JCP (Java community Process). Le Java Community Process regroupe les entreprises suivantes : Sun, IBM, Oracle, Borland, Nokia, Sony, la fondation Apache, ObjectWeb...

4. L'architecture Java EE en couche

4.1. Introduction

L'architecture classique d'une application Java EE est organisée en couches, chacune ayant un rôle bien défini. Considérons l'architecture proposée par SUN:

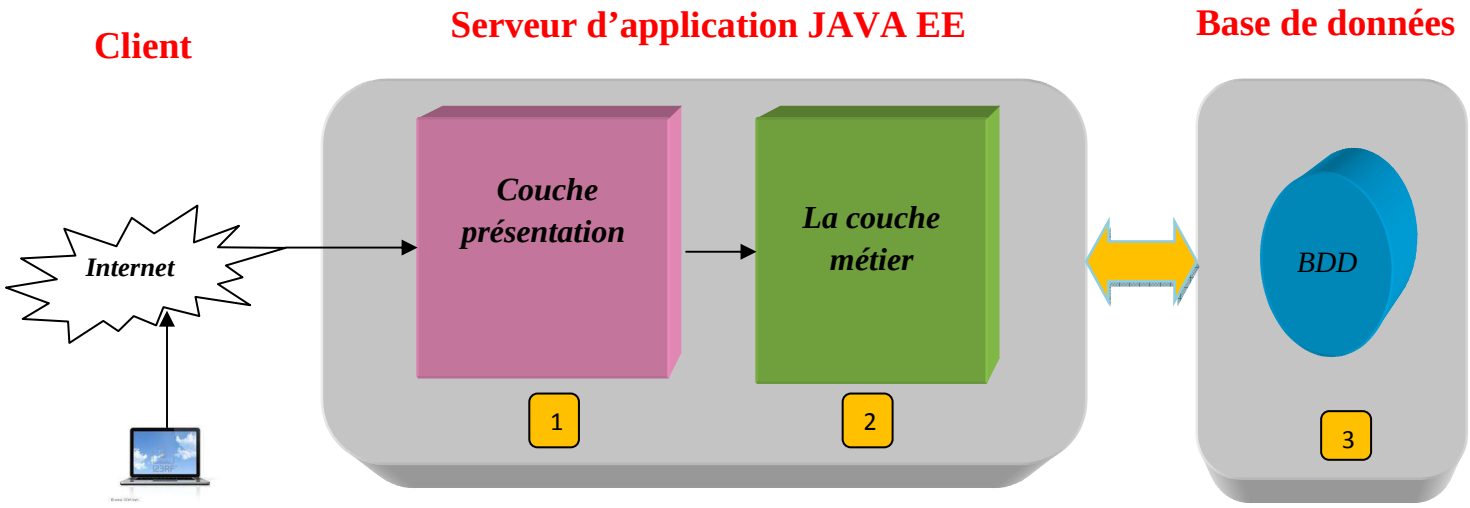
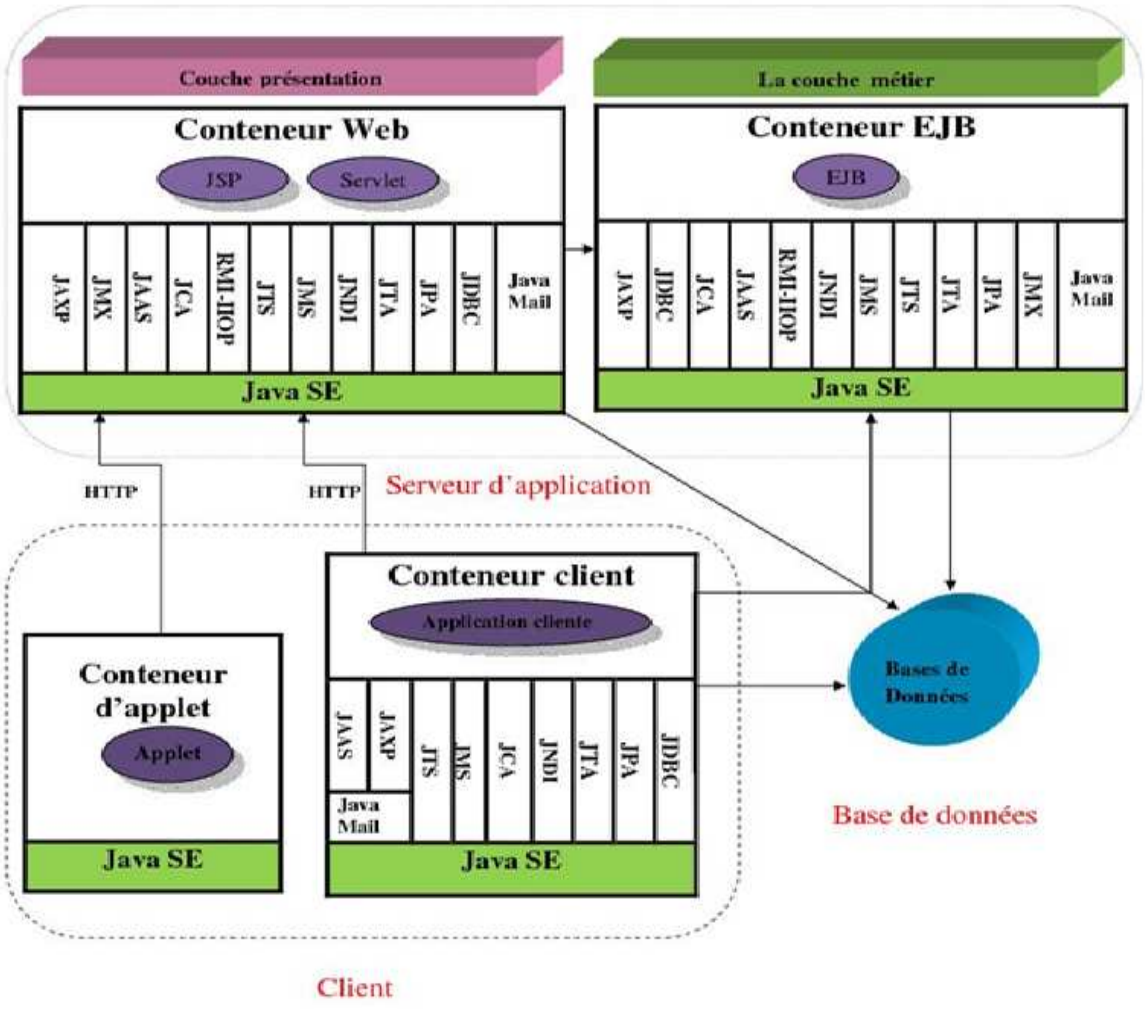


Figure II.1. *Architecture Java EE en couche*

- *La couche [1] est la couche qui dialogue avec l'utilisateur (client), via une interface graphique, une interface console ou une interface web. Elle a pour rôle de fournir des données provenant de l'utilisateur à la couche [2] ou bien de présenter à l'utilisateur des données fournies par la couche [2].*
- *La couche [2] est la couche métier qui se compose de 3 parties qui sont :*
  la logique métier, les Objets d'accès aux données et les objets de données.
- *La couche [3] est la couche base de données qui inclut la base de données.*

4.2. Schéma général de l'architecture Java EE



4.2.1.Composants Java EE

- *Un composant est une unité logicielle de niveau applicatif.*
- *En plus des JavaBeans, qui font partie du J2SE, Java EE supporte les types de composants suivants :*
  - *Applets.*
  - *Application clientes.*

- *Composants Enterprise JavaBeans (EJB).*
- *Composants Web.*

- *Les applets et applications clientes sont exécutées sur le poste du client tandis que les composants EJB et Web fonctionnent sur le serveur.*

4.2.2. Le conteneur de composants Java EE

Les conteneurs assurent la gestion du cycle de vie des composants qui s'exécutent en eux. Les conteneurs fournissent des services qui peuvent être utilisés par les applications lors de leur exécution.

4.2.3. Client Java EE

La plateforme J2EE prévoit que plusieurs types de clients puissent accéder à une même application et interagir avec les composants côté serveur.

4.2.4. Applets

- *Les Applets sont des clients Java qui s'exécutent habituellement dans un navigateur web et qui ont accès à toutes les possibilités du langage Java.*
- *Les applications Java EE peuvent utiliser des clients applets pour avoir des interfaces utilisateurs plus puissantes que celles connues en HTML.*
- *Les applets communiquent avec le serveur par HTTP.*

4.2.5. Application cliente

- ✓ *Des applications clientes s'exécutent dans leur propre conteneur client.*
- ✓ *Le conteneur client est un jeu de librairies et d'API qui supporte le code client, et met à sa disposition un moyen d'accéder aux composants métiers de l'application.*
- ✓ *Les applications clientes ont des interfaces utilisateurs qui peuvent directement interagir avec le tier EJB en utilisant RMI-IIOP.*
- ✓ *Les clients ont un accès complet aux services de la plate-forme Java EE. Comme les services de nommage JNDI, l'envoi de messages et JDBC.*
- ✓ *Le conteneur client gére l'accès à ces services et les communications RMI-IIOP.*

Remarques :

1. *Une API est un ensemble de fonctions, procédures ou classes mises à disposition par une bibliothèque logicielle, un système d'exploitation ou un service. La connaissance des API est indispensable à l'interopérabilité entre les composants logiciels.*

Les API peuvent être regroupées en trois grandes catégories :

- *Les composants : Servlet, JSP, EJB.*
- *Les services : JDBC, JTA/JTS, JNDI, JCA, JAAS.*
- *La communication : RMI-IIOP, JMS, Java Mail.*

2. *L'ensemble des API sera expliqué en détail dans la suite de ce chapitre.*

4.3. Couche présentation

La couche de présentation assure la logique de navigation mais aussi la gestion des droits de l'utilisateur (authentification, droits). Elle se compose en règle générale de composants de type Servlets et JSP.

Les servlets serviront pour le contrôle, c'est-à-dire qu'elles doivent exécuter l'ensemble des traitements liés aux requêtes utilisateurs mais également les appels aux modèles métiers.

Les JSP, quant à elles, seront utiles pour la représentation graphique des informations et elles ne doivent pas êtres utilisés à la place du contrôleur.

Par exemple, si vous souhaitez traiter un formulaire, vous utiliserez une servlet plutôt qu'une page JSP. Les JSP doivent avoir qu'un minimum de code Java afin d'être optimale.

Cependant les Servlet/JSP étant des composants de bas niveau, elles n'offrent que des fonctionnalités de base. Une des solutions est d'utiliser des Framework. Le plus connu est sans doute Struts. Ce Framework utilise le concept des Servlet / JSP mais ajoute l'ensemble de l'architecture MVC afin de simplifier la vie du développeur.

Suite à Struts, les JSF (Java Server Faces) ont vu le jour et risquent de s'afficher comme un standard d'ici peu de temps et donc de remplacer Struts. En effet, JSF est intégré dans la plate-forme Java EE.

4.3.1. Les Servlets

Définition:

Les servlets sont des composants logiciels entièrement écrits en Java, elles effectuent des traitements côté serveur en réponse aux requêtes des clients distants. L'API Servlet fournit des élements nécessaires à la conception des composants web dynamiques.

4.3.2. Les JSPs

Définition

La JSP est une extension de la technologie Java Servlet qui permet de programmer simplement l'affichage de contenus dynamiques sur le Web. JSP consiste en une page HTML incluant du code Java qui s'exécutera soit sur le serveur Web, soit sur le serveur d'application. Le langage HTML décrit la manière dont s'affiche la page, le code Java servant à effectuer un traitement, par exemple récupérer les informations nécessaires pour effectuer une requête dans une base de données.

4.4. La couche métier

Elle se décompose en 3 parties :

La couche métier

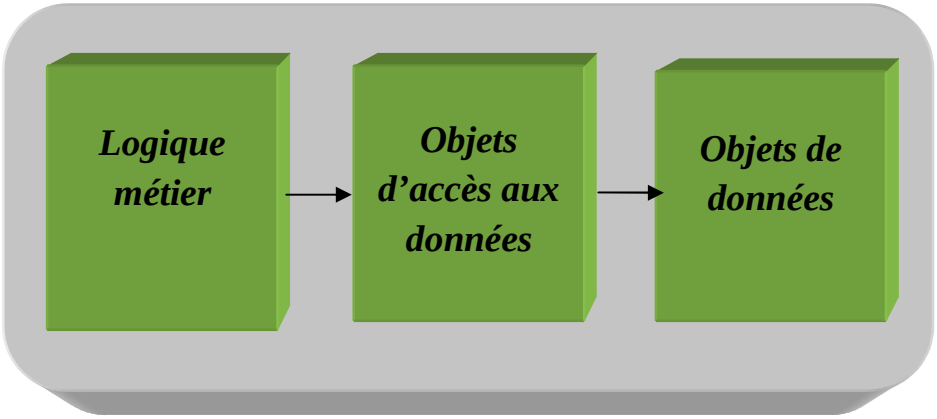


Figure II.3 *La couche métier*

4.4.1. Logique métier

Elle correspond à la partie fonctionnelle de l'application, celle qui implémente la « logique spécifique de l'application», et qui décrit les opérations que l'application opère sur les données en fonction des requêtes des utilisateurs, effectuées à travers de la couche présentation.

Les différentes règles de gestion et de contrôle du système sont mises en œuvre dans cette couche.

4.4.2. Accès aux données

Elle consiste en la partie gérant l'accès aux données du système. Ces données peuvent être propres au système, ou gérées par un autre système. Cette couche permet à la couche logique métier d'accéder aux données d'une manière uniforme, celle-ci n'a pas à s'adapter aux changements possibles dans les couches supérieures puisque toutes les opérations sont maintenues transparentes par la couche d'accès aux données.

4.4.3. Objets de données

Elle assure la persistance des données, à chaque table de la base de données correspond une classe entité.

- ***Les EJB***

1.Déscription

Les Enterprise JavaBean sont des composants Java portables, réutilisables et déployables qui peuvent être assemblés pour créer des applications. Un EJB est accompagné d'un ou plusieurs fichiers de déploiement écrit en XML qui permet au serveur d'application de déployer correctement l'objet au sein d'un conteneur.

- *Les classes Java doivent respecter certaines règles et fournir certaines méthodes.*
- *Le composant EJB s'exécute dans un conteneur EJB, qui prend en charge tout ce qui concerne le niveau système.*
- *Répartition des tâches entre le programmeur de bean et le conteneur.*
- *Les EJB ne sont qu'une spécification.*

- *Les composants EJB fonctionnent avec tout type de client :*
  - *servlets et JSP.*
  - *clients Java (via RMI).*
  - *clients et serveur divers (via RMI/IIOP).*
  - *... etc.*

2.Types d'EJB

2.1.EJB Entity

C'est une représentation orientée objet de données stockées dans la base de données.

- *Les clients peuvent y accéder en toute sécurité simultanément.*
- *La durée de vie du bean est exactement identique à celle des données qu'il représente.*

Ils représentent les objets du domaine, et plus précisément ceux qui ont besoin d'être rendus persistants. Ils ont pour objectifs de fournir au développeur la possibilité de manipuler directement un modèle de composants métier. Ce modèle est, quant à lui, mappé sur le schéma d'une base de données. Ce type d'EJB s'occupe des données.

2.2. EJB Session

Un EJB session est un EJB de service dont la durée de vie correspond à un échange avec un client. Ils modélisent le processus métier de l'application (traitement).

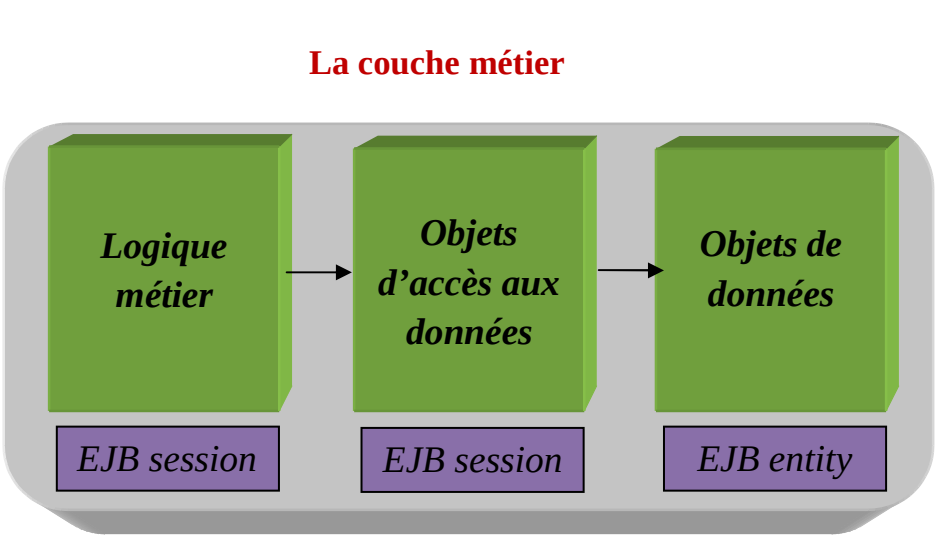


Figure II.4 *L'emplacement des EJBs dans la couche métier.*

4.5. Couche BDD

Inclut la base de données.

5. Le serveur d'application Java EE

5.1. Définition

Les serveurs d'application sont des logiciels qui aident des entreprises à développer, déployer et contrôler un grand nombre d'applications qui sont la plupart du temps distribuées. Du point de vue développement, la différence principale apportée par un serveur d'application est la séparation de la logique métier de la logique présentation et de la logique base de données. Essentiellement, les serveurs d'application nous aident à démontrer des applications 3-tiers où la base de données est logiquement séparée (parfois physiquement séparé aussi) de la logique métier.

Un serveur d'application peut simplifier votre procédé de développement. Les serveurs d'application prennent habituellement soin de la plupart, sinon de toutes les questions techniques impliquées, et permettent aux développeurs de se concentrer sur la raison première du projet. Ceci permet de budgéter pour des systèmes beaucoup plus grands et beaucoup plus utiles.

5.2. Les conteneurs

Les conteneurs fournissent les supports d'exécution pour les composants d'applications Java EE. Ces conteneurs fournissent aux composants applicatifs une vue agrégée des toutes les APIs Java EE. Les composants applicatifs Java EE n'interagissent jamais directement avec d'autres composants applicatifs. Ils utilisent les protocoles et méthodes des conteneurs pour interagir les uns avec les autres et avec les services des plates-formes. Cette introduction d'un médiateur entre les composants et les services Java EE permet aux conteneurs d'injecter de manière transparente les services définis par les descripteurs de déploiement, tels que le management déclaratif de transaction, les vérifications de sécurité... etc.

Un serveur d'application Java EE contient deux conteneurs : le conteneur Web et le conteneur EJB.

5.2.1. Le conteneur Web

Le conteneur web est un logiciel qui exécute les servlets et les JSP et assure leur cycle de vie. Il fournit des services qui peuvent être utilisés par les applications lors de leur exécution.

5.2.2. Le conteneur EJB

La définition du conteneur EJB est similaire à celle du conteneur web la différence est que les conteneurs EJB exécutent les EJB.

Remarque :

Il y a chevauchement entre un serveur d'application et un serveur web, car tous les deux peuvent exécuter des taches. Le serveur web (serveur HTTP) peut appeler des scriptes et des services pour interroger des bases de données et pour exécuter la compilation ; tandis que les serveurs d'application viennent, très souvent avec leur propre serveur de HTTP qui fournit des pages Web au navigateur.

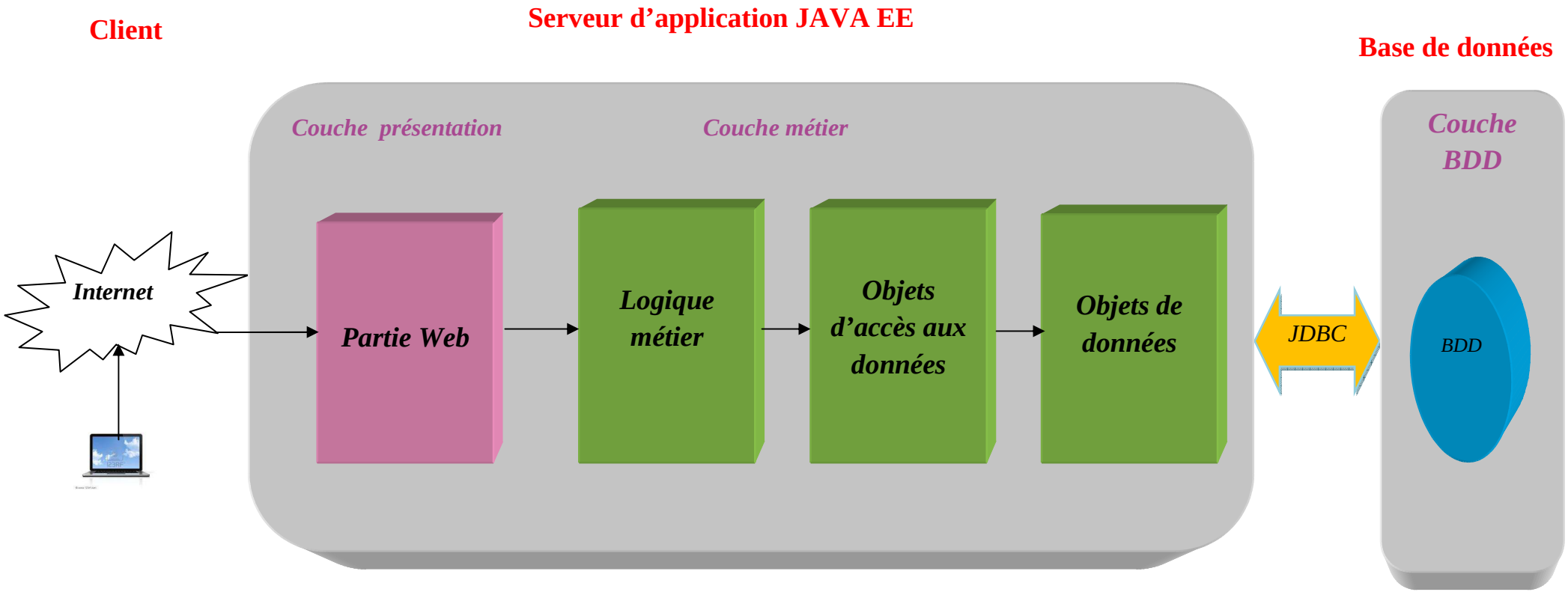


Figure II.5. *Schéma détaillé de l'architecture Java EE en couche*

5.3. Les avantages d'un serveur d'application

Les serveurs d'application possèdent un grands nombre d'avantages :

- ***Client léger :***

Fournit un client beaucoup plus léger. Même si vous avez des codes source de Java que vous distribuez aux utilisateurs, il est très petit.

- ***Intégrité de données et du code :***

En centralisant la logique métier sur un ou sur quelque machines serveur, des mises à jour et des mises à niveau des applications pour tous les utilisateurs peuvent être garanties.

- ***Configuration centralisée***

Des changements de configuration d'application, telle qu'un mouvement des serveurs de base de données, peuvent être faits de manière centralisé.

- ***Sécurité***

Un point central par lequel l'accès aux données ou parties de l'application elle même peut être contrôlé est considéré comme un avantage de sécurité, plaçant la responsabilité de l'authentification loin de la couche potentiellement peu sûre de client sans exposer la couche de base de données.

- ***Passage à l'échelle***

En limitant le trafic de réseau au trafic de couche présentation, le modèle client/server améliore l'exécution de grosses applications dans les environnements d'utilisation intensive.

5.4. Quelques serveurs d'application Java

Glassfish, BEA Weblogic, JBoss, IBM Webspher, J2EE x, JOnAS, Tomcat ...etc.

5.5. Les APIs

J2EE regroupe un ensemble d'API pour le développement d'applications d'entreprise :

API	***Rôle***
J2EE Connector Architecture (JCA)	*Connecteurs pour accéder à des ressources du système d'information de l'entreprise telles que CICS, TUXEDO, SAP ...*
Remote Method Invocation (RMI) et RMI-IIOP	*RMI permet l'utilisation d'objets Java distribués. RMI-IIOP est une extension de RMI pour une utilisation avec CORBA.*
Java Naming and Directory Interface (JNDI)	*Accès aux services de nommage et aux annuaires d'entreprises*
Java Database Connectivity (JDBC)	*Accès aux bases de données. J2EE intègre une extension de cette API*
Java Transaction API (JTA) Java Transaction Service (JTS)	*Support des transactions.*

Java Messaging service (JMS)	*Support de messages via des MOM (Messages Oriented Middleware)*
Java Server Faces (JSF)	*Facilite et standardise développement d'applications Web.*
Java Authentication and Authorization Service (JAAS)	*Echange sécurisé de données*
Java Persistance API (JPA)	*Gestion de la persistance des données*
JavaMail	*Envoi et réception d'e-mails*
Java Management Extensions (JMX)	*Extension d'administration des applications*
Java API for XML Parsing (JAXP)	*Analyse et exploitation de données au format XML*

Conclusion

Dans ce chapitre nous nous somme familiarisés à fur et à mesure a la technologie Java EE. Nous nous somme rendus compte qu'elle est adéquate a notre travail car elle rassemble a la fois les avantages d'une plate-forme, de Java et ceux de l'architecture trois tiers.

Notre travail s'est consacré ensuite a l'étude de l'architecture Java EE en couche. Celle-ci se décompose en trois parties essentielles :

- *Couche présentation*
- *Couche métier*
- *Couche Base de Données*

Nous nous somme intéressés aux serveurs d'application Java EE et les différentes APIs qu'ils intègrent.

A la fin nous avons présenté le schéma de l'architecture Java EE utilisé dans l'organisme d'accueil.

Annexe III

Design Pattern

- Définition
- Représentation / identification
- Classification des patterns
- Design patterns en J2EE**Erreur ! Signet non défini.**
- Le MVC
- Les patterns et la reconstruction logicielle

Sommaire

Liste des figures

Introduction

La programmation orientée objet fait son apparition dans les années 60, et depuis, celle-ci n'a cessée de prendre du terrain, et est devenu manifestement incontournable. La POO veut que l'on éclate les applications en composants simples (voir atomique) et réutilisables.

Néanmoins, si ceci s'effectue sans règles précises, le concepteur finira par être submergé par l'immensité et complexité du codage. Pour y remédier il a fallu exploiter un niveau plus haut dans la conception objet.

*Le concept de **design patterns** est né en 1977, des travaux de l'architecte Christopher Alexander, qui remarqua que la phase de conception en architecture laisse apparaître des problèmes récurrents. Il cherche alors à résoudre l'ensemble de ces problèmes liés à des contraintes interdépendantes. Il établit un langage de 253 patterns, qui couvrent tous les aspects de la construction.*

*Le concept est développé dans un ouvrage, intitulé '**Design Patterns : Elements of Reusable Object-Oriented Software**' publié en 1995 par le '**Gang of Four**'. Il présente 23 Design Patterns qui font aujourd'hui référence dans le monde de l'informatique. Bien plus tard, deux livres s'inspirant du GoF mais se spécialisant à la plate-forme J2EE font leurs apparitions. « EJB Design Pattern » de Floyd Marinescu et « Core J2EE Pattern » de Deepak Alur, John Crupi et Dan Malks.*

1. Définition

Design patterns, formes, modèles ou bien patrons de conception. Décrivent, formalisent des solutions éprouvées à des problèmes spécifiques et récurrents. « Un patron décrit un problème devant être résolu, une solution, et le contexte dans lequel cette solution est considérée. Il nomme une technique et décrit ses coûts et ses avantages » [Johnson 1997].

2. Représentation / identification

Le GoF a adopté une représentation uniforme et structurée. Chaque patron est décrit et documenté de la même façon. Les propriétés utilisées sont :

Nom, Intention, Alias, Motivation, Indications d'utilisation, Structure, Participants, Collaborations, Conséquences, Implémentation, Exemple de code, Utilisations connues et Patrons apparentés.

Cela dit, il s'est avéré, au cours de nos recherches, que la plus part des articles se restreignait à n'en citer que : le nom, la problématique, la solution et les conséquences d'utilisations.

3. Classification des patterns

Plusieurs travaux, autre que ceux du GoF, se sont intéressés à l'étude et à la documentation des patterns. Cependant, en ce qui concerne les patrons de conception, le travail du GoF a énormément contribué à leur émergence. On se limitera, donc, à la description du formalisme utilisé par le GoF.

Le classement des design patterns s'est effectué selon leur rôle et leur domaine d'application. Il en dégagera trois catégories, énumérées ci-dessous.

3.1. Patterns relationnels

Concernent la création de classes ou d'objets. Ils sont au nombre de cinq :

- ***Abstarct factory*** *(Fabrique abstraite)*
- ***Builder*** *(Monteur)*

- ***Factory method*** *(Fabrique)*
- ***Prototype*** *(Prototype)*
- ***Singleton*** *(Instance inique).*

3.2. Patterns structurels

S'intéressent à la composition d'objets ou classes pour réaliser de nouvelles fonctionnalités. Ils sont au nombre de sept :

- ***Adapter*** *(Adaptateur)*
- ***Bridge*** *(Pont)*
- ***Composite*** *(Objet composite)*
- ***Decorator*** *(Décorateur)*
- ***Façade*** *(Façade)*
- ***Proxy*** *(Proxy)*
- ***Flyweight*** *(Poids-plume)*

3.3. Patterns comportementaux

Concernent les interactions entre classes et l'affection des responsabilités.

- ***Chain of responsibility*** *(Chaîne de responsabilité)*
- ***Command*** *(Commande)*
- ***Interpreter*** *(Interpréteur)*
- ***Iterator*** *(Itérateur)*
- ***Mediator*** *(Médiateur)*
- ***Memento*** *(Mémento)*
- ***Observer*** *(Observateur)*
- ***State*** *(État)*
- ***Strategy*** *(Stratégie)*
- ***Template Method*** *(Patron de méthode)*
- ***Visitor*** *(Visiteur)*

4. Design patterns en J2EE

Vue que l'on travail sous la plate-forme J2EE, on est contraint de solliciter, en plus de ceux de base, les patterns utilisés au sein de celle-ci.

Il y avait, initialement, quinze (15) patterns, identifiés par le Sun java Center, utilisés au niveau des trois couches (3 tiers). Des nouveau patterns voient le jour en repense à de nouveau besoin. On citera ci-dessous les cinq patterns les plus utilisés :

4.1. Session Façade

Chaque appel distant est coûteux, il faut trouver un moyen de les limiter. Une Session Façade est un session bean qui a accès aux interfaces locales d'autres beans. Un appel à une méthode d'un Session Façade entraîne généralement plusieurs appels vers un ou plusieurs autres beans. Au lieu de faire plusieurs appels

(ex : chercher un espace, copier, coller, supprimer l'originale), il permet d'en faire qu'un pour arriver au même résultat (déplacer). Il porte bien son nom, un mur entre application et persistance dans le bute de limier les appels distants. Sans doute le pattern J2EE le plus utilisé.

4.2. Value Object

Le Session Façade implique un challenge supplémentaire. Puisque nous n'avons pas de référence directe à notre objet persistent, comment allons-nous lire le contenu d'un objet ? Une solution très simple, et qui ajoute même de nombreuses possibilités d'amélioration. Le Value Object n'est autre qu'un objet qui offre une vue figée de notre entité à un moment donné.

4.3. Service Locator

Le client GUI, SC ou autre a besoin d'un moyen de localiser les objets distants, en l'occurrence notre Session Façade. Le service de nommage couramment utilisé avec J2EE est JNDI (méthode supposée connue). Malgré tout, dans le but de rendre notre système flexible et réutilisable, nous définissons un Service Locator, dont le seul but est de servir au client les références des objets distants.

4.4. Data Access Object

Au besoin d'exploiter un système persistant de données, il peut y avoir beaucoup de mécanismes différents et donc beaucoup de différences entre les API utilisées pour accéder aux données.

*Difficile et couteux à mettre en place, **Data Access Object** propose d'uniformiser l'accès aux données sur différents systèmes de stockage (BDD relationnelle, un annuaire LDAP, des documents XML, etc.) en n'exposant que les interfaces simples permettant de manipuler les données.*

4.5. Business Delegate

Ce pattern n'est autre qu'une adaptation du pattern Adapter, appliqué au modèle J2EE. Son but est multiple :

- *Se débarrasser de la dépendance du client pour J2EE. J2EE offre les bases nécessaires pour séparer le client du serveur. Complètement. Dès lors, un « import javax.ejb.* » dans une servlet est malvenu. Pourquoi ? Parce que le développeur de votre EJB peut changer certaines signatures de méthodes à tout moment. Cela veut-il dire que chacune de vos servlets doit être mise à jour ? Le Business Delegate va nous permettre de regrouper en un seul endroit tous les appels distants.*
- *Transformer les Exceptions génériques en Exceptions propres à votre application.*
- *Proposer des méthodes spécifiques à votre client.*
- *Exécuter des portions de code nécessaires à chaque appel distant.*

4.6. Annexe design Patterns J2EE

Deux patterns œuvrent coté requêtes :

- ***Intercepting Filter*** *(Filtre d'interception)*
- ***Front Controller*** *(Front contrôleurs)*

Et voici le reste des patterns :

- ***Service to Worker*** *(service des travailleurs)*

- ***Transfer Object Assembler*** *(Assembleur d'objet de transfert)*
- ***Value List Handler*** *(Manutention Liste Valeur)*
- ***Composite Entity*** *(Entité Composite)*
- ***Service Activator*** *(Actionneur)*
- ***Composite View*** *(Vue composite)*
- ***View Helper*** *(Aide de vue)*
- ***Dispatcher View*** *(Répartition)*

5. Le MVC

Créé par Xerox PARC pour Smalltalk-80, le Model-View-Controller est un pattern très répandu et fort utile (recommandé en J2EE). Il est née du désire de séparer les données, les traitements et la présentation. Ainsi l'application se retrouve segmentée en trois composants essentiels : le modèle, la vue et le contrôleur, comme le montre la figure 1 suivante.

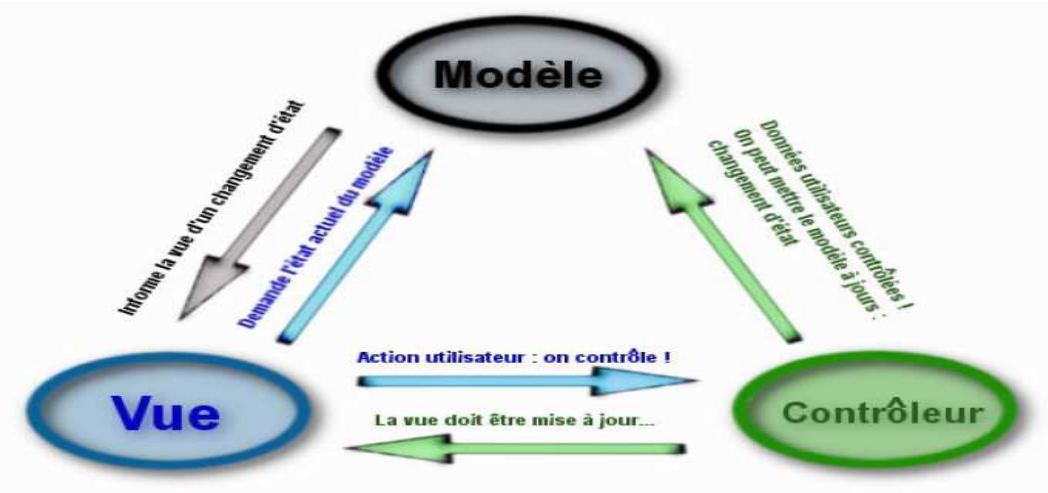


Figure III.1 : *Schéma de représentation/fonctionnement du pattern MVC*

5.1. Le modèle

Représente les données et les règles métiers. C'est dans ce composant que s'effectuent les traitements liés au cœur du métier. Les données peuvent être liées à une base de données, des EJBs, des services Web, … Il est important de noter que les données sont indépendantes de la présentation. En d'autres termes, le modèle ne réalise aucune mise en forme. Ces données pourront être affichées par plusieurs vues. Du coup le code du modèle est factorisé.

5.2. La vue

Correspond à l'IHM. Elle présente les données et interagit avec l'utilisateur. Dans le cadre des applications Web, il s'agit d'une interface HTML, mais n'importe quel composant graphique peut jouer ce rôle.

5.3. Le contrôleur

Se charge d'intercepter les requêtes de l'utilisateur, d'appeler le modèle puis de rediriger vers la vue adéquate. Il ne doit faire aucun traitement. Il ne fait que de l'interception et de la redirection.

5.4. A propos de MVC

Le MVC favorise le développement et la maintenance du code. Sur de gros projets et/ou avec de grandes équipes de développements, l'application d'un tel modèle de conception se révèle très performante. Il existe aujourd'hui des Frameworks très avancés qui se basent sur le MVC. L'utilisation de ces Frameworks facilite sa mise en place et cadre sa réalisation.

6. Les patterns et la reconstruction logicielle

La gestion de configuration de logicielle (SMC) est une discipline du génie logiciel ayant pour objet de répondre à la question : On a mis au point (construit) un produit (logiciel), Comment le reproduire efficacement et d'une manière plus rapide ? Le plus souvent, il ne s'agit pas de reproduire à l'identique, mais de reproduire avec des modifications incrémentales. Voici quelques un de ces patterns:

- *Bill of Materials*
- *Self-Identifying Configuration*
- *Reproducible Build*
- *Daily Build and Smoke Test*
- *Shared Version Cache*
- *Shared Object Cache*
- *Shared Source Escalation*
- *Version-Controlled Environment*
- *Archived Environment*

Les patterns utilisés dans le cadre de la reconstruction logiciel sont des intermédiaires entre le nouveau produit (celui qu'on veut reconstruire) et les versions précédente de ce produit. Ils seront utiles ultérieurement. On prend l'exemple du ***Bill of Materials****, qui est une liste de tous les composants qui ont contribué à la construction, il peut contenir les noms, les versions et les chemins de répertoire des systèmes d'exploitation, bibliothèques, compilateurs, éditeurs de liens ...etc.*

Conclusion

Nous avons fait le tour des design patterns, dans cette section. Allant de ceux de base, du ***GoF,*** *jusqu'à ceux utilisé dans le cadre de la reconstruction logiciel, en passant par ceux s'adaptant à la plate-forme J2EE, qui sont les plus importants nous concernant, entre autres le fameux MVC très utilisé en J2EE. Il en existe beaucoup plus que ceux qu'on a vu et on serra certainement amené a en voire de nouveaux.*

Ce qui nos motive à en utiliser, c'est leur aspect générique, considérés comme des micros architectures, ils visent à réduire la complexité, à promouvoir la réutilisation et à fournir un vocabulaire commun aux concepteurs.

De leur apparition à nos jours, le développement d'application se voit de plus en plus performant et tend à la perfection. Ils ont, en génie logiciel, une place des plus importantes, qui tire ca grande valeur des nombreux objectifs atteint par leurs utilisations, chose promise chose due.

Ouvrages

- Documentation interne de Marine Soft : rapports d'ingénieurs.
- L'approche processus, mode d'emploi, Jean-Pierre WOJTYNA, Hans BRANDEBOURG, 2009
- Rapport sur les progiciels Marine Soft Direction Technique, Département Recherche & Développement, Septembre 2011.
- Architecture J2EE, Patrick ITEY, Juin 2011.
- UML2, pratique de la modélisation 2ème édition, Benoit CHARROUX, Aomar OSMANI, Yann THIERRY-MIEG, 20011.
- Rapport sur les outils de travail,, Marine Soft Direction Technique, Département Recherche & Développement, Décembre 2011.
- The unified software développement process, Olivier GERBE, 2010
- Programmeur Java SE 6, Campus Press, publié en Juillet 2010.
- Développement Web : « Zone grand débutants», Publié en 2010, dernière mise à jour le 16 janvier 2011.
- Introduction à JSF « Java Server Faces », Publié par la boite « **ATOL Conseils et Développements** 3 bd Eiffel 21600 LONGVIC » en 2009.
- RicheFaces Guide du développeur, Martin HELLER, publié le 05/05/2010.
- Cours JPQL, publié par loic FRERING et Baptiste MEURANT, le 16/12/2008.
- FAQ BIRT. Cette FAQ a été réalisée par l'ensemble des membres du forum Business Intelligence. Les personnes contribuées à cette FAQ : Bérénice MAUREL - Stefan CARACAS - Julien SAUVEBOIS- GATTINO – Charly CLAIRMONT - Erwan BODERE, Publié le : 11/03/2009.
- JBoss Application Server : exploitation et sécurisation, Renaud DUBOURGUAIS le 17/12/2006.
- Cour Plugin Eclipse, Pierre-Arnaud MARCELOT 2.0 Laurent, AUDIBERT février 2007.
- UML par la pratique, Etudes des cas et exercices corrigés, Pascal ROQUES, Avril 2001.
- Design Pattern Elements of Reusable Object-Oriented Software, Erich GAMMA, Richard HELM, Ralph JOHNSON, John VLISSIDES, Addison WESLEY. 1995.
- Comprendre les différents design patterns de construction par Jean-Michel Ormes, 2011.

- Design Patterns, O. BOISSIER, G. PICARD, SMA/G2I/ENS Mines Saint-Etienne, Septembre 2009.
- Design Patterns, University of Paderborn, Software Engineering Group, 2009.
- Introduction aux Design Patterns en Java, Alexandre Brillant, 2011.
- Réutilisation dans les SI : patrons de conception, Yannick PRIE, SIMA, 2008.
- Détection automatique des patrons de conception, M. HERMAK, L. EL BADAOUI, université de Montréal, 2008.
- Refactoring des applications Java/J2EE, Jean - Philippe RETAILLE, 2011.
- Patterns J2EE / EJB, Christophe LUDET, Mars 2005.
- Design Pattern and Software Architecture, Dr. Giese HOLGER, University of Paderborn, Software Engineering Group, 2011.
- Les patrons de conception: Représentation et mise en œuvre, G. EL BOUSAIDI H. MILLI, LTCE, Université du Québec à Montréal, 2008.
- Patron de conception, Wikipédia, Février 2012.
- Les designs patterns ou Les motifs de conception ou Les modèles de conception, JEE Pour Tous, Mai 2011.
- Mise en œuvre du design pattern "Business Delegate, Alexis HASLER, 2009
- Le rapport JEE FINAL réalisé par MARINE SOFT en 10/09/2009.
- Java EE Guide de développement d'applications Web en Java réalisé par Mr Jérôme LAFOSSE, 2007
- Introduction à Java EE 5 avec Netbeans 6.8 et le serveur d'applications Glassfish V3, Mr Serge TAHE à istia.univ-angers.fr juin 2010.
- L'architecture J2EE, ludovic.maitre@free.fr, 2012
- Informatique Répartie Introduction aux EJB, Alexandre PAUCHET, 2011
- LES SERVEURS D'APPLICATION, Elysée FONTEP, Phuong-Anh HOANG, 2011
- Mémoire d'ingénieur « Conception et réalisation d'un site web interactif pour la vente de véhicules Toyota ». Cas « concessionnaire Toyota Haddad ». Réalisé par : M^{elle} BERKAL Farida & M^{r} BENYAHIA Ramdane. Proposé et dirigé par : M^{r} Samir HAMEG. Promotion 2004/2005.
- Mémoire d'ingénieur « Extension et adaptation d'un système d'information à la mobilité ». Cas «suivi pédagogique ». Réalisé et présenté par : M^{r} Lyes BOUMGHAR & M^{r} Ali HAOUCHINE. Proposé et dirigé par : M^{me} M.BELKADI. Promotion 2008/2009.

Les sites web

http://laurent-piechocki.developpez.com/uml/tutoriel/lp/cours/

http://www.marinesoft.dz/

http://vogella.developpez.com/tutoriels/eclipse/prise-en-main-eclipse-ganymede/

http://www.itechno.com/fr/index.php

http://www.betecommechou.com/2009/02/j2ee-1-4-composants-applicatifs-et-conteneurs

http://jmdoudoux.fr/java/dej/chap-j2ee-javaee.htm