REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Mouloud Mammeri de Tizi-Ouzou

FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

Mémoire

De fin d'études
En vue de l'obtention du diplôme de
Master en Informatique
OPTION : Systèmes informatiques

Thème

PATRONS DE CONCEPTION POUR L'INTEGRATION DE SYSTEME

Proposé et dirigé par :
Mr KERBICHE MHAND

Présenté par :
M^{elle} CHABNI Nawel
M^{elle} SAIDJ Sabrina

PROMOTION : 2012/2013

REMERCIEMENTS

Nos vifs remerciements pour notre promoteur *M^r^ KERBICHE Mhand* de nous avoir proposé ce thème et de nous l'avoir éclairé, comme nous tenons à le remercier pour ses précieux conseils et à sa disponibilité pendant la durée de ce travail.

Comme nous tenons à remercier l'ensemble des enseignants qui nous ont suivies durant notre cursus.

Sans oublier de remercier nos familles et nos amis qui nous ont toujours soutenues et encouragées.

Dédicaces

Je dédie ce modeste travail à :

Mes très chers parents : Arezki et Houria

Mes chers frères : Said, Yacine, Rachid et Amine

Mes adorables sœurs : Lynda, Lila, Kahina et Sonia

A mes beaux frères : Arezki et Rezki

Aux anges : Ghiles, Ilane et Sofiane

A mon oncle Belaid et toute sa famille

A ma grand-mère maternelle Ouardia

A tous mes amis (es)

A mon binôme Sabrina Et toute sa famille

A tous ceux qui me connaissent

Nawel

Dédicaces

Je dédie ce modeste travail à :

Mes très chers parents : Meziane et Taklit

Mes chers frères : Farid et Samir

Mes adorables sœurs : Samira, Farida et Hassina

A mes belles sœurs : Naima et Saliha

A mes beaux frères : Malek, Khaled et Salim

A ma nièce : Melissa

A mes neveux : Mohand, Lounes, Yacine, Tahar , Said et Islam

A toute ma famille

A tous ceux qui m'ont aidé de prés ou de loin dans mes études

A mes très chers amis (es)

A tous mes proches grands et petits

A mon binôme Nawel pour sa patience

Et toute sa famille

Sabrina

Sommaire

Chapitre II : Intégration et interopérabilité

Chapitre III : Analyse et conception

Chapitre IV : Réalisation

Liste des figures

Introduction générale

Les systèmes d'information (SI) sont une brique essentielle à l'organisation de nos entreprises et plus généralement de notre société. Ils sont aujourd'hui construits à partir de l'intégration de systèmes informatiques qu'il convient de maintenir et faire évoluer avec agilité et sans entropie. Les objectifs sont alors entre autres d'améliorer la qualité des services offerts tout en préservant l'autonomie des acteurs, l'ouverture des SI, une gestion cohérente des informations, des temps de production réduits et une meilleure maitrise des coûts de maintenance.

L'accès à l'information et l'échange de données sont des enjeux fondamentaux de l'entreprise. La garantie d'un environnement informatique disponible, fiable, sécurisé, performant et adapté devient un avantage capital.

Les patrons de conception (en anglais design patterns), en caractérisant des problèmes récurrents de conception et en spécifiant des solutions claires et élégantes à ces problèmes, représentent le vocabulaire commun de l'expertise des concepteurs de logiciels. Ce vocabulaire leur fournit un niveau de description adéquat pour discuter les choix de conception ou de restructuration d'un système, au-delà des détails connus de conception. De plus, l'explicitation des patrons existants dans un système simplifie la compréhension de sa structure.

Certains travaux de recherche proposent des techniques d'implémentation facilitant la réutilisation des patrons. En général, ces outils ont pour principal avantage d'améliorer la traçabilité de la présence de patrons dans les logiciels réalisés. Pour ce faire, ils imposent pour chacun une réalisation particulière (par exemple sous la forme d'une classe). Ils ne prennent donc pas en compte la pluralité des variantes d'implémentation qui fait la richesse de la notion de patron, ce qui limite fortement leur champ d'application.

En effet, l'intérêt d'un patron est de regrouper sous une étiquette unique une variété de situations implémentatoires qui ont « quelque chose en commun ». Sa mise en œuvre est pilotée par différents compromis, et peut adopter différentes formes ou variations, sans pour autant déroger à la solution de conception qu'il définit.

L'interopérabilité est un moyen d'assurer l'intégration. La différence entre intégration et interopérabilité est clarifiée par le standard ISO 14258 (1999) qui considère que des modèles peuvent faciliter : l'intégration quand il existe un standard ou une représentation pivot qui les formalise ; l'unification quand il existe un méta-modèle commun assurant une équivalence sémantique entre eux ; la fédération lorsque les modèles existent par eux-mêmes mais que des correspondances entre les concepts qu'ils modélisent peuvent être définis à un niveau ontologique pour formaliser la sémantique de leur interopérabilité.

L'intégration est souvent considérée comme allant plus loin que l'interopérabilité, en forçant une certaine dépendance fonctionnelle des applications. Alors que des systèmes interopérables peuvent fonctionner indépendamment, un système intégré peut perdre certaines

fonctionnalités si certains de ses services sont interrompus. Un système intégré est donc composé d'applications interopérables mais des applications interopérables ne réalisent pas nécessairement un système intégré.

En complément de ces deux vues systémiques, certains systèmes peuvent n'être que compatibles. Ces systèmes n'interfèrent pas directement entre eux. Cela n'implique donc pas qu'ils soient capables d'échanger des services. Des systèmes interopérables sont, par définition, compatibles, au moins en partie, mais l'inverse ne l'est pas.

❖ **Problématique :**

Comment redonner l'accès à des systèmes complexes plus faciles et plus fonctionnels en gardant ses performances tout comme un seul système cohérent ?

❖ **Objectifs :**

Il est essentiel de comprendre les techniques bien éprouvées, qui ont déjà montré leur capacité à résoudre des problèmes de développement récurrents. Les Patrons de Conception («Design Patterns») sont des canevas («frameworks») qui aident à saisir, spécifier et mettre en œuvre ces techniques éprouvées.

Les développeurs de logiciel se confrontent à des problèmes qui sont largement indépendants de l'application elle-même.

Les bons développeurs résolvent ces problèmes en s'appuyant sur les patrons de conception.

Pour garantir une meilleure intégration, le patron de conception doit :

- Résoudre un problème récurrent qui correspond à une solution éprouvée.
- Favoriser l'extensibilité.
- Améliorer la flexibilité et la réutilisabilité
- Aider au développement de logiciels par la réutilisation de l'expérience collective des ingénieurs expérimentés en informatique.
- Aider à promouvoir les bonnes pratiques de conception, en capturant les expériences existantes et bien validées en développement logiciel.
- Aider à la gestion de la complexité du logiciel.
- Faciliter la communication entre les développeurs.
- Faire inter opérer des systèmes déjà en service.

La suite de ce mémoire est structurée en quatre chapitres :

Afin de mieux comprendre l'enchaînement des concepts fondamentaux de notre travail, le premier chapitre est un résumé d'un de ces concepts en définissant la notion de patron de conception, nous scindons cette notion en deux contextes différents : patron et patron de

conception ; dont on donne quelques exemples de patron tel que le patron classe, ancêtre, conteneur, utilisateur et messager ; et aussi quelques patrons de conception tel que le patron de conception prouvé, Gof ; ainsi les catégories de ces derniers.

Le deuxième chapitre est consacré pour l'intégration et l'interopérabilité, deux concepts fondamentaux de notre travail, on s'intéresse au premier en donnant sa définition ainsi que ses différents types et puisque le deuxième concept est un moyen pour assurer le premier, alors on la définit en donnant ces principes ainsi que ses niveaux.

Dans le troisième chapitre, nous avons un cas d'étude à analyser et faire dégager quelques patrons de conception déjà vus au premier chapitre tel que adaptateur, façade, pont et fabrique abstraite et les étudier un à un en donnant le problème traité et faire ressortir les solutions possibles en suite.

Implémenter ces patrons de conception en utilisant Oracle 11g, PL/SQL Developer version 7.0.0.1050 et NetBeans IDE 7.2.1 sera notre objectif dans le dernier chapitre.

CHAPITRE I

PATRONS DE CONCEPTION

Introduction :

En génie logiciel, un patron de conception (design pattern) est un concept destiné à résoudre les problèmes récurrents suivant le paradigme objet.

Les patrons de conception décrivent des solutions standards pour répondre à des problèmes d'architecture et de conception des logiciels. À la différence d'un algorithme qui s'attache à décrire d'une manière formelle comment résoudre un problème particulier, les patrons de conception décrivent des procédés de conception généraux. On peut considérer un patron de conception comme une formalisation de bonnes pratiques, ce qui signifie qu'on privilégie les solutions éprouvées.

Il ne s'agit pas de fragments de code, puisque les patrons de conception sont le plus souvent indépendants du langage de programmation, mais d'une méthode de conception, c'est-à-dire d'une manière standardisée de résoudre un problème qui s'est déjà posé par le passé. Le concept de patron de conception a donc une grande influence sur l'architecture logicielle d'un système.

On peut donc considérer les patrons de conception comme un outil de capitalisation de l'expérience appliqué à la conception logicielle.

I.1. Les patrons (Patterns):

I .1.1. Définition de patron (Pattern) : [1] [2]

Un patron décrit à la fois un problème qui se produit très fréquemment dans l'environnement et l'architecture de la solution à ce problème de telle façon que l'on puisse utiliser cette solution des milliers de fois sans jamais l'adapter deux fois de la même manière.

Appelés également "motifs", ils représentent des modèles permettant de résoudre des problèmes de modélisation, à différentes échelles. En général, un patron possède quatre éléments essentiels :

- Un nom qui est un moyen de décrire en un ou deux mots un problème, ses solutions et leurs conséquences.
- Un problème qui décrit les situations où le modèle s'applique. Il expose le sujet à traiter et son contexte.
- Une solution qui décrit les éléments pour résoudre le problème, les relations entre eux, leurs parts dans la solution, et leur coopération.
- Des conséquences qui sont les effets résultants de la mise en œuvre du modèle et les compromis que cela entraîne.

I.1.2. Types de Patterns : [3]

- **Architectural Patterns :** Un patron d'architecture est un modèle de référence qui sert de source d'inspiration lors de la conception de l'architecture d'un système informatique.
- **Design Patterns :** Est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels. Les patrons de conception décrivent des procédés de conception généraux et permettent en conséquence de capitaliser l'expérience appliquée à la conception de logiciels. Ils ont une influence sur l'architecture logicielle d'un système informatique.
- **Idioms ou coding patterns :** sert à lier une solution à un langage particulier.
- **Anti-patterns :** Les anti-patrons ou anti-pattern sont des erreurs courantes de conception des logiciels. Leur nom vient du fait que ces erreurs apparaissent dès les phases de conception du logiciel, notamment par l'absence ou la mauvaise utilisation de patrons de conception. Les anti-patrons se caractérisent souvent par une lenteur excessive du logiciel, des coûts de réalisation ou de maintenance élevés, des comportements anormaux et la présence de bugs.
- **Organizational patterns :** Modes d'organisation sont des structures de parenté, généralement à une organisation professionnelle, qui aident l'organisme à atteindre ses objectifs. Les motifs sont généralement inspirés par l'analyse de plusieurs organisations professionnelles et de trouver des structures communes dans leurs réseaux sociaux . Ces modèles sont collectés et organisés en langues de motifs, qui sont publiés en tant que fondation pour l'amélioration des processus et la conception organisationnelle, en grande partie dans la communauté de développement de logiciels.

I.1.3. Exemples de patrons :

I.1.3.1. Le patron classe (pseudo classe) : [4]

Pseudo classe est en fait le patron de base à partir duquel tout autre patron peut être défini lorsqu'on donne une définition générale d'une classe.

Seuls les exemples de classes sont bien des classes. Une difficulté majeure rencontrée par les concepteurs débutants est celle d'utiliser correctement le patron Classe.

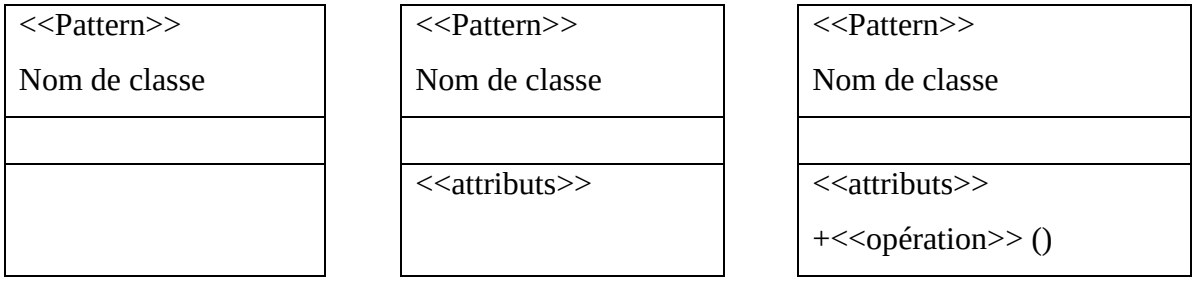

Figure I.1 : patron Classe

Le patron classe pour être <<bien>> utilisé doit introduire une cohérence forte entre le nom de la classe et la liste des noms des méthodes de la classe. Il en est de même pour les attributs.

Pour éviter de se focaliser sur les données internes d'un objet, il est essentiel de porter son attention sur l'interface d'utilisation de la classe.

Règle 1 : Toute classe doit implémenter au moins une interface (ensemble d'opérations publiques) cohérente avec le nom de la classe (donc avec le concept ou la chose représentée par cette classe). Une classe est un modèle de serveur de service.

Règle 2 : Les attributs d'une classe doivent être considérés comme privés. Même les méthodes d'accès en lectures des attributs d'une classe ne doivent être utilisées qu'à titre exceptionnel (justification à donner) sauf si ces attributs sont des objets à part entière.

Règle 3 : Une classe doit représenter un seul concept et tout ce concept (cohésion d'une classe).

I.1.3.2. Le patron Ancêtre : [4]

Le patron ancêtre exprime la possibilité d'étendre les possibilités d'une classe en utilisant la relation d'héritage.

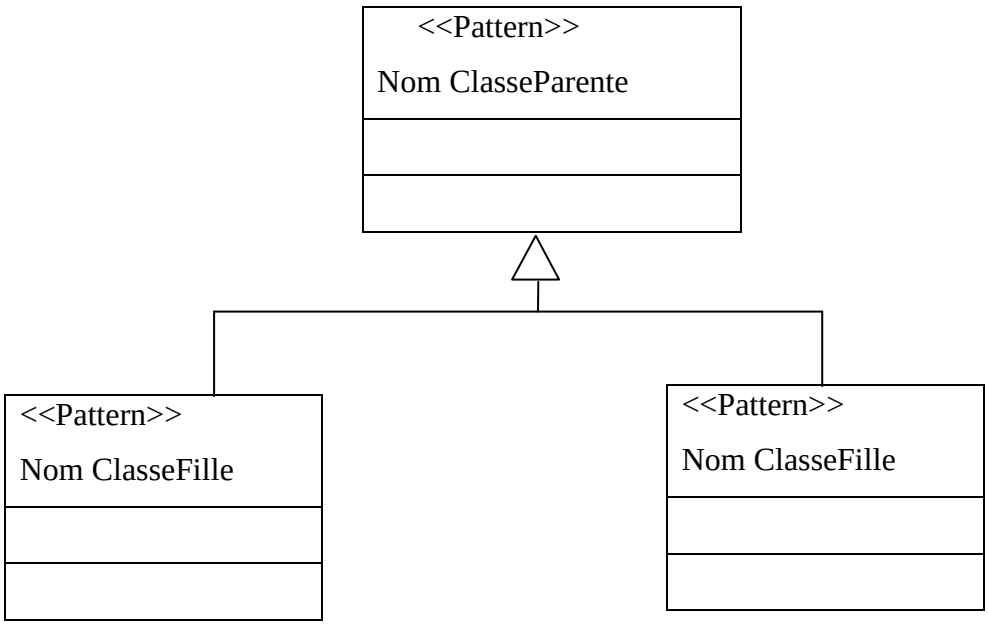

Figure I.2 : Patron Ancêtre

Une sous classe ou classe descendante est conceptuellement de la même nature que la classe ancêtre. Les sous classes d'une même classe ancêtre sont aussi conceptuellement proches puisqu'elles appartiennent à la même famille. Une classe ancêtre ne voit pas à priori ses classes descendantes. En revanche, une classe descendante voit toutes ses classes ancêtres mais pas ses collatérales. De plus l'affectation entre classes collatérales n'est pas autorisée. La mutation entre classes apparentées est possible en utilisant le patron de passage d'une sous classe à une autre.

- **Utilisation**

Les classes descendantes d'un même niveau d'une classe ancêtre représentent des déclinations de la classe ancêtre. Une sous classe constitue un aspect ou un point de vue de la classe racine. Le patron ancêtre forme une hiérarchie d'abstraction (de la classe la plus abstraite aux classes les plus spécialisées).

La relation ancêtre descendante n'est pas toujours appropriée pour représenter un concept donné. En particulier l'utilisation de l'héritage pour effectuer une partition résultant d'une dichotomie sur plusieurs propriétés n'est pas intéressante. Dans ces situations l'utilisation du patron conteneur sera plus adéquate.

Règle 4 : Pour des raisons pratiques, toute classe est la descendante d'au plus une classe ancêtre (héritage simple) obligation dans le cas de java.

Règle 5 : Toutes les classes descendantes immédiates d'une classe ancêtre possèdent (au moins) une méthode ayant la même signature (condition de polymorphisme).

Règle 6 : Toutes les méthodes définies au niveau des classes ancêtres doivent avoir un sens lorsqu'elles sont appliquées à des instances d'une classe descendante. Dans certains cas particuliers (à éviter), cela peut ne pas être vérifié. Dans ces cas, il est impératif d'intercepter le message non significatif au niveau de la classe descendante en levant une exception.

Règle 7 : Un message envoyé à une instance d'une hiérarchie classe ancêtre- classes descendantes est traité soit :

- Par la classe de l'objet ;
- Par une classe ascendante (parente) ;
- Par la classe de l'objet puis complété par une classe ancêtre ;
- Dans tous les autres cas par la classe ancêtre racine.

Règle 8 : Le nom d'un objet instance d'une classe descendante doit être construit à partir du nom d'une classe ascendante.

Règle 9 : On peut effectuer une instance d'une classe descendante à une instance d'une classe ascendante, mais la réciprocité est interdite. L'affectation entre classes collatérales est également interdite.

I.1.3.3. Le patron conteneur : [4]

Le patron conteneur met en œuvre un couple de patrons classes reliés par une relation de composition par valeur (agrégation ou inclusion).

Figure I.3 : Patron conteneur

Les noms de la classe parente et de la classe fille doivent pouvoir s'énoncer sous la forme suivante : un objet de la classe fille est une composante de la classe parente (ou la classe parente est composée de la classe fille). Une partie du nom de la méthode de la classe fille peut (ou doit) se trouver dans l'interface de la classe parente. On dit que la classe parente « encapsule » ses classes filles.

Construire un objet d'une classe parente indique la construction des objets des classes filles. La destruction des classes filles n'est pas autorisée sans passer par la destruction de l'objet conteneur.

Une classe composante peut être la composante de plusieurs classes « conteneur ». Au niveau des instances un objet composant n'appartient qu'à un seul conteneur. Une classe ne peut pas être une composante d'elle-même (ni introduire un cycle de fermeture transitive : C1 contient C2,…et Cn qui contient C1). Une composante peut être optionnelle ou multiple.

- **Utilisation :**

Le patron conteneur est un élément d'abstraction très puissant. Il doit être utilisé toutes les fois ou il est naturel de regrouper les composantes entre elles pour les abstraire en un concept plus général. Les détails et la complexité d'un objet sont cachés aux utilisateurs de cet objet. L'objet est manipulé comme un tout sans avoir à se préoccuper de ce qu'il contient.

Règle 10 : Au niveau de la conception (modèle logique) tous les attributs atomiques d'une classe sont considères comme des instances du patron conteneur. Les types prédéfinis de base sont considérés comme des classes prédéfinis.

Règle11 : Lorsque plusieurs composantes sont de la même classe, la racine des noms d'attributs est commune et la fin est particulière à la composante.

I.1.3.4. Le patron utilisateur : [4]

Le patron utilisateur est constitué de deux pseudos classes reliées par une relation d'utilisation (de visibilité ou de délégation).

Figure I.4 : Patron utilisateur

Une méthode d'une instance de la classe utilisatrice (objet utilisateur) envoie un message à une instance de la classe utilisée (objet utilisé), ce qui nous place dans les situations suivantes :

Un objet est soit : une variable globale, une variable locale d'une méthode de l'objet utilisateur, un paramètre formel d'une méthode de l'objet utilisateur, un objet retourné par une

méthode de l'objet utilisé, un attribut(ou champ) de l'objet utilisateur (et qui est aussi dans ce cas utilisé), l'objet lui –même.

Règle 12 : L'usage de variables globales est prohibé. Donc, tout envoi de message est assuré de la présence du receveur puisque le destinataire est nécessairement dans la portée de visibilité de l'émetteur.

Règle 13 : Les responsabilités de construction d'un objet en conséquence d'une délégation sont :

-Une variable locale est construite dans la méthode de l'objet utilisateur, la méthode sait facilement si l'objet existe ou pas ;

-Les objets attributs d'un objet conteneur sont construits par le constructeur d'objet de la classe « méthode de classe » du conteneur. Les méthodes de la classe conteneur font confiance au constructeur d'objet qui doit assurer l'existence des objets composants.

-Un paramètre effectif (à l'envoi d'un message) est soit une variable locale, soit un paramètre de la méthode, soit l'objet émetteur lui- même (self), soit un attribut de l'objet utilisateur. Une méthode suppose que les paramétrés sont correctement passés.

-L'objet retourné est soit une variable locale, soit un paramètre passé en entrée, soit l'objet utilisé lui –même (self), soit un attribut de l'objet utilisé.

Règle 14 : Un objet passé en paramètre d'un message est partagé par les objets utilisateur, est utilisé (passage par référence). On peut dupliquer un objet par un clonage, mais le passage s'effectuera lui-même par référence. Il est de la responsabilité de l'utilisateur d'effectuer une copie par valeur de l'objet si les modifications de l'objet ne doivent pas être répercutées sur l'objet original.

- **Utilisation :**

Un envoi de message à un objet déclenche une méthode au niveau de l'objet. Cet objet peut avoir besoin, pour effectuer le service demandé, des services d'autres objets. Il est du ressort de la conception de décider si les services doivent être créés par l'objet (cas de variable locale), passés en paramètres du message ou encore se trouver dans les champs de la classe de l'objet. La méthode déclenchée décidera, lors de l'invocation de nouveaux messages, d'effectuer l'appel avec un passage de paramètre ou non.

Si un patron est à la fois utilisateur et conteneur alors les objets composants de l'objet utilisateur ne voient pas l'objet utilisé. Si un objet conteneur souhaite faire utiliser à une de ses composantes l'objet utilisé, il est nécessaire de le passer en paramètre.

Règle 15 : Les signatures doivent être le plus simple possible (moins de paramètres possibles, un objet auquel on s'adresse connait tous ses champs), sans pour cela avoir recours à des

variables globales. L'utilisation de paramètres typés (chaine, entier, caractère…..) au détriment d'objets doit être si possible évitée.

Règle 16 : Le patron utilisateur remplace un lien d'association binaire entre les deux classes lorsque l'association correspond à une action placée sur l'une ou l'autre des classes. Un lien réciproque (symétrique de l'association) correspond alors à un appel avec passage de l'objet lui-même.

I.1.3.5. Patron Messager : [4]

Dans une démarche objet, les objets ne représentent pas uniquement une structure de données. On s'adresse à un objet pour obtenir un service. Les services d'un objet sont en rapport avec les données internes de l'objet. Ainsi lorsqu'on souhaite transmettre un objet à un autre objet, la première solution est de passer en paramètre une méthode de l'objet destinataire. Cependant, si plusieurs cas de manipulation de l'objet « émis » sont à considérer, le code spécifique va se trouver dans la méthode de l'objet récepteur, alors que ce code devrait se trouver au niveau de l'objet transmis. Pour faire en sorte que l'objet transféré soit responsable des traitements le manipulant, il suffit de s'adresser à l'objet lui-même en lui passant en paramètre l'objet destinataire, à moins qu'il ne soit « connu » de l'objet à émettre.

Pour traiter de manière générale la transmission d'un objet à un autre, il faut utiliser le patron Ancêtre pour l'objet à émettre (chaque sous-classe implémente un cas spécifique d'émission) ainsi que pour l'objet receveur (chaque classe implémente un cas spécifique de réception). Enfin, le patron Messager résulte de l'utilisation du patron utilisateur entre les deux racines des patrons ancêtres.

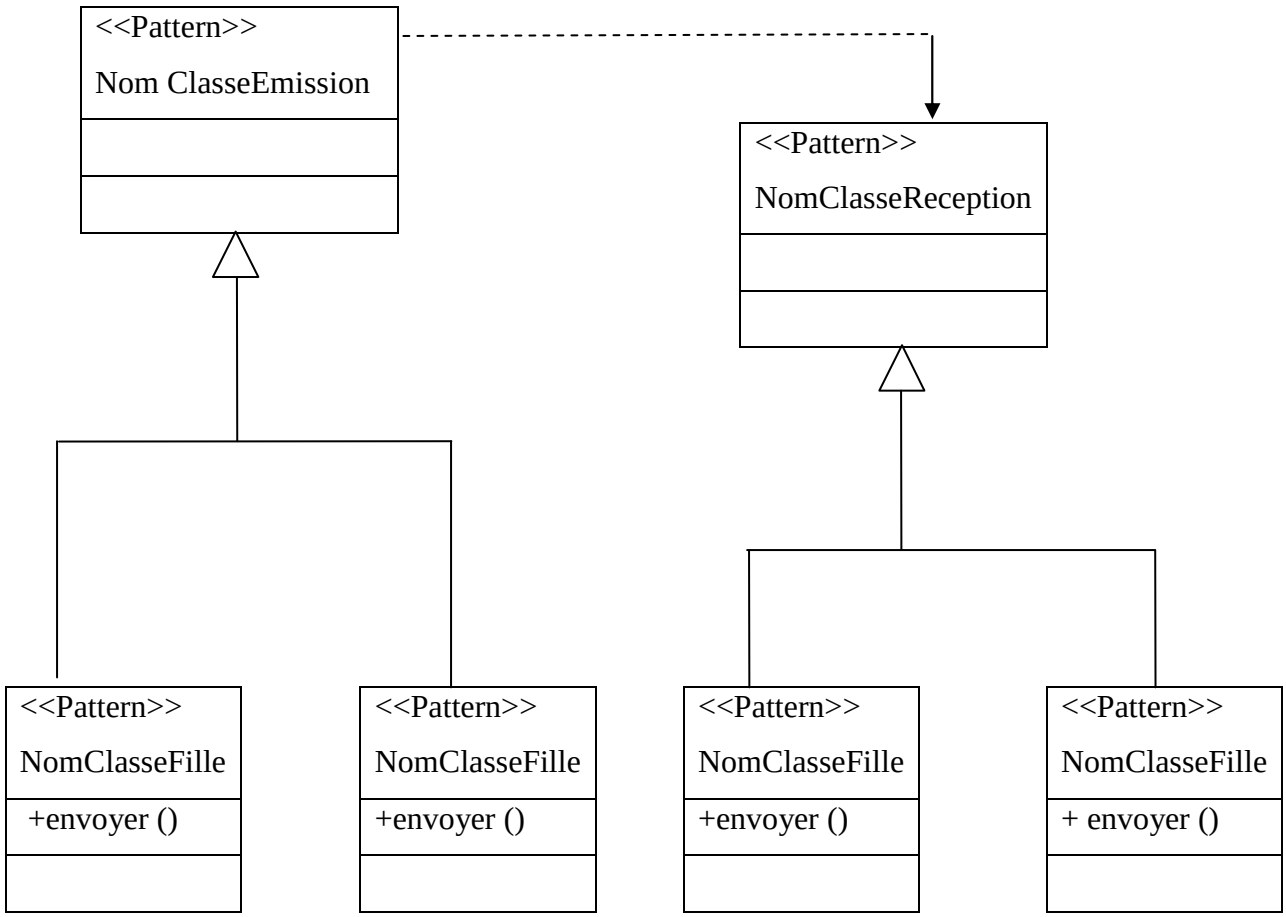

Figure I.5 : Patron Messager

Classiquement si on désire regrouper les traitements spécifiques à une donnée particulière on utilisera des noms de procédures différentes. Lorsque le langage le permet il est possible de faire de la surcharge de noms (même nom de procédure, mais les types ou le nombre de paramètres sont distincts). Il est également possible d'utiliser une seule procédure avec une instruction de type « case of » pour traiter chacune des situations particulières. Dans une démarche objet on ne devait jamais (ou presque) se poser de questions sur la nature de l'objet auquel on s'adresse. En utilisant le polymorphisme au travers du patron Ancêtre nous effectuons la transmission de l'objet en laissant au concepteur la possibilité d'ajouter ultérieurement d'autres modes d'envoi ou de réception sans avoir à modifier le code des classes existences. Toutes les sous-classes doivent comporter la redéfinition de méthode « envoyer » ou « transmettre » pour un destinataire implicite. Si c'est nécessaire on ajoutera à chaque sous classe de l'objet émis la méthode « envoyer A » ou « transmettre A » dans le cas d'un destinataire explicite passé en paramètre.

- **Utilisation :**

Tout objet construit à partir du patron Messager devient à la fois le messager et le message à transmettre. Quelle que soit sa nature, il suffit de lui adresser soit le message « envoyer » si le destinataire est connu par le messager (dépendant d'un contexte), soit le message « envoyer A » en passant l'objet destinataire en paramètre.

Règle 17 : Tout objet émis suivant le patron Messager doit répondre aux messages « envoyer » ou « envoyer A » qui lui sont envoyés.

Règle 18 : Tout objet récepteur suivant le patron Messager doit « répondre » au message « envoyer ».

I.1.4. Framework : [4]

Un Framework est un pattern architectural qui fournit un canevas extensible aux applications d'un domaine. Un Framework est plus grand qu'un patron de conception (mécanisme). En fait, on peut présenter un Framework comme une sorte de micro architecture qui comprend plusieurs mécanismes travaillant ensemble pour résoudre un problème récurrent dans un domaine donné. Lorsqu'on définit un Framework, on précise l'armature d'une architecture, ainsi que les connecteurs, les onglets, les boutons et les cadrans que l'on expose aux utilisateurs qui souhaitent adapter ce Framework à leur propre contexte.

En UML, on modélise un Framework comme un paquetage stéréotypé. Si on détaille ce paquetage, on découvre des patrons de conception qui résident dans de nombreuses vues d'architecture d'un système.

I.1.4.1. Techniques de modélisation :

I.1.4.1.1. Modélisation de patrons de conception : [4]

Lors de la modélisation d'un pattern de conception (design pattern), on doit prendre en compte sa vue interne et sa vue externe.

Vu de l'extérieur, un pattern de conception est présenté par une collaboration paramétrée. Comme une collaboration, un pattern fournit un ensemble d'abstractions dont la structure et le comportement coopèrent pour exécuter une fonction utile. Les paramètres de la collaboration désignent les éléments qu'un utilisateur de ce pattern doit lier. Le pattern de conception devient alors un canevas qu'on utilise dans un contexte particulier en fournissant des éléments qui correspondent aux paramètres du canevas.

Vu de l'intérieur, un pattern de conception est simplement une collaboration et est représenté avec ses parties structurelles et comportementales. En général, on modélise l'intérieur de cette collaboration à l'aide d'un ensemble d'interactions (pour l'aspect comportemental). Les

paramètres de la collaboration désignent certains de ces éléments structurels, qui lorsque le pattern de conception est rattaché à un contexte particulier, sont instanciés à l'aide de l'abstraction tirée de ce contexte. Pour modéliser un pattern de conception, il faut :

- Identifier la solution courante du problème récurrent et la réifier comme un mécanisme ;
- Modéliser le mécanisme comme une collaboration en fournissant ses aspects structurels et comportementaux ;
- Identifier les éléments du pattern de conception qui doivent être rattachés à des éléments dans un contexte spécifique et les représenter comme des paramètres de la collaboration.

I.1.4.1.2. Modélisation des patterns architecturaux : [4]

La modélisation de patterns architecturaux est l'autre raison pour laquelle on utilise des patterns. En effet, lorsqu'on modélise un tel Framework, on modélise l'infrastructure d'une architecture complète que l'on réutilise et d'adapter à un certain contexte.

On représente un Framework par un paquetage stéréotypé. Tout comme un paquetage, un Framework fournit un ensemble d'éléments, parmi lesquels se trouvent (liste non exhaustive) des classes, des interfaces, des cas d'utilisation, des composants, des nœuds, des collaborations et même d'autres Frameworks. Certains de ces éléments sont publics et représentent des ressources sur lesquelles les clients peuvent s'appuyer. Ce sont les « onglets » du Framework que l'on peut connecter aux abstractions du contexte. Certains de ces éléments publics sont des patterns de conception et représentent des ressources que les clients utilisent. Enfin, certains de ces éléments sont protégés ou privés et représentent des éléments encapsulés du Framework, visibles uniquement de l'intérieur.

Lorsqu'on modélise un pattern architectural, il ne faut pas oublier un Framework est en fait une description d'une architecture, bien qu'elle soit incomplète et probablement paramétrée. Par conséquant, tout ce que l'on sait sur la modélisation d'une architecture bien structurée s'applique à la modélisation de Frameworks bien structurés.

Pour modéliser un pattern architectural, il faut :

- Extraire le Framework d'une architecture existante et éprouvée ;
- Modéliser le Framework comme un paquetage stéréotypé qui contient tous les éléments (et en particulier les patterns de conception) nécessaires et suffisants pour décrire les différentes vues de ce Framawork.
- Exposer les connecteurs, les onglets, les boutons et les cadrans nécessaires à l'adaptation du Framework sous la forme de patterns de conception et de collaboration. Pour

l'essentiel, cela signifie qu'il est nécessaire que l'utilisateur sache clairement quelles classes doivent être étendues, quelles opérations doivent être implantées et quels signaux doivent être manipulés.

I.2. Patron de conception (Design pattern) :

I.2.1. Définition : [6]

Les design patterns apportent des solutions à des problèmes communs de conception de logiciels. Dans le cas de la programmation orientée objet , design patterns sont généralement destinées à résoudre les problèmes de génération d'objets et d'interaction, plutôt que sur les problèmes à plus grande échelle de l'architecture globale du logiciel. Ils donnent des solutions généralisées sous la forme de modèles qui peuvent être appliquées aux problèmes du monde réel.

Les design patterns sont un outil puissant pour les développeurs de logiciels. Toutefois, ils ne devraient pas être considérés comme des spécifications normatives pour les logiciels. Il est plus important de comprendre les concepts qui décrivent les modèles de conception, plutôt que de mémoriser leur position exacte des classes , méthodes et propriétés . Il est également important d'appliquer des motifs de manière appropriée. En utilisant le schéma incorrect pour une situation ou d'appliquer un modèle de conception à une solution triviale peut compliquer votre code et conduire à des problèmes de maintenabilité.

I.2.2.Patron de conception prouvé : [7]

Premiers patrons identifiés d'après le cas d'étude de la presse (INRS).C'est une notion proposée par J.R. Abrial, consiste à réutiliser des résultats de modélisation qui ont été prouvés. Un patron de conception n'est considéré comme « prouvé » qu'une fois qu'il a été utilisé avec succès au moins dans trois cas.

I.2.3.Pourquoi définir des patrons de conception ? [8]

- Construire des systèmes plus extensibles, plus robustes au changement
- Capitaliser l'*expérience collective* des informaticiens
- Réutiliser les solutions qui ont fait leur preuve
- Identifier les avantages/inconvénients/limites de ces solutions
- Savoir quand les appliquer

I.2.4.Choix d'un patron de conception : [8]

- Est-il une solution au problème ?
- Quels sont ses buts ?
- Quelles sont les relations avec les autres patrons de conception ?
- Est-ce que d'autres patrons jouent le même rôle ?

- Mise en œuvre d'un patron de conception
- Lire complètement la description, en particulier les sections *indications d'utilisation* et *conséquences*.
- Étudier en détail les sections *Structure*, *Constituants* et *Collaborations*
- Regarder la section *Exemple de code*
- Choisir des noms de constituants ayant un sens dans le contexte d'utilisation !

I.2.5. Mise en œuvre d'un patron de conception : [8]

- Lire complètement la description, en particulier les sections indications d'utilisation et conséquences.
- Étudier en détail les sections Structure, Constituants et Collaborations
- Regarder la section Exemple de code
- Choisir des noms de constituants ayant un sens dans le contexte d'utilisation !

I.2.6. Ce qu'il ne faut pas attendre des patrons de Conception : [8]

- Une solution universelle prête à l'emploi
- Une bibliothèque de classes réutilisables
- L'automatisation totale de l'instanciation d'un patron de conception
- La disparition du facteur humain

I.2.7. Les problèmes d'utilisation et d'application des patrons : [9]

Les patrons de conception ne sont pas tous au même niveau de complexité. En effet, l'application, de certains d'entre eux, n'est pas un processus évident. Il faut cerner le problème pour pouvoir identifier la solution adéquate et donc le patron adéquat. Une fois que le patron à utiliser est identifié, il faut l'appliquer au modèle concerné. Pour cela, le concepteur doit pouvoir définir le rôle de chaque élément du modèle pour établir une correspondance avec les éléments du patron.

Une fois le patron appliqué, il faudra vérifier si la sémantique du modèle est toujours respectée, comme il faudra s'assurer que de futures modifications ne violent pas les contraintes sémantiques et structurelles imposées par le patron.

L'implémentation des patrons dans un langage de programmation nécessite une mise en correspondance entre les éléments de conception du patron et les constructions du langage de programmation. Cela n'est pas toujours aussi évident, dans le cas du langage Java, par exemple, on ne peut pas implémenter l'héritage multiple.

I.2.8. Description des patrons par GoF : [4] [9]

Le *Gang of Four* sont les auteurs du livre « Design Patterns: Elements of Reusable Object-Oriented Software » édité en 1995. Cet important ouvrage décrit les techniques de développement différents et des pièges en plus de fournir vingt-trois orientés objet design patterns de programmation. Les quatre auteurs étaient Erich Gamma , Richard Helm , Ralph Johnson et John Vlissides .

Ils proposent de solutions élégantes, et toujours différentes pour résoudre des différents problèmes récurrents rencontrés par les architectes logiciels.

GoF ont adopté une représentation uniforme et structurée. Chaque patron est décrit et documenté de la même façon.

Les propriétés utilisées sont :

_ **Nom :** Le nom significatif du patron.

_ **Intention :** Décrit ce que fait le patron, son but, quel problème particulier il résout.

_ **Alias :** Enumère, s'il en existe, d'autres noms communs du patron.

_ **Motivation :** Donne un exemple de problème de conception pouvant être résolu par application du patron. L'illustration par un exemple facilite la compréhension de la description abstraite (donnée par la suite) du patron.

_ **Indications d'utilisation :** Décrit les situations où on peut utiliser le patron.

_ **Structure :** Décrit la structure du patron en utilisant des diagrammes de classes. Des diagrammes de collaboration sont aussi utilisés pour la description de l'interaction entre les classes du patron.

_ **Participants :** Définit les classes (ou objets) intervenantes dans le patron et leurs responsabilités.

_ **Collaborations :** Décrit comment les participants interagissent pour assumer leurs responsabilités.

_ **Conséquences :** Décrit comment le patron atteint ses objectifs, quels sont les compromis et les résultats de son utilisation.

_ **Implémentation :** Présente des astuces, techniques et pièges qu'il faut connaître lors de l'implémentation du patron. Cette partie propose aussi, quand il en existe, des solutions typiques du langage utilisé.

_ **Exemple de code** : Donne des fragments de code pour illustrer la façon dont on peut implémenter le patron en C++ ou Smalltalk.

_ **Utilisations connues :** Contient des exemples d'utilisation du patron dans des systèmes réels.

_ Patrons apparentés : Cite les patrons qui sont reliés avec celui en cours de traitement et leurs différences.

I.2.9. Types de patrons de conception selon GoF : [4]

GoF ont classé les patrons qu'ils proposent selon leur rôle et leur domaine d'application (classe vs objet). Ils ont distingué entre patrons créateurs, structurels et comportementaux. Les patrons créateurs concernent la création de classes ou d'objets. Les patrons structurels s'intéressent à la composition d'objets ou classes pour réaliser de nouvelles fonctionnalités. Finalement, les patrons comportementaux concernent les interactions entre classes et l'affection des responsabilités.

On présentera dans ce chapitre les Design patterns, recueil connu de vingt trois patrons de GoF qui sont très utilisés en conception objet.

		Catégorie		
		Créateur	Structurel	Comportemental
Portée	Classe	Fabrication	Adaptateur (classe)	Interprète
				Patron de méthode
	Objet	Fabrique abstraite	Adaptateur (objet)	Interprète
		Monteur	Pont	Commande
		Prototype	Composite	Iterateur
		Singleton	Décorateur	Médiateur
			Façade	Mémento
			Poids mouche	Observateur
			Procuration	Etat
				Stratégie
				Visiteur

Figure I.6 : Catalogue GoF Patterns

I.2.9.1) Les patrons créateurs : [1] [2]

Les patrons créateurs proposent des solutions pour gérer l'instanciation d'objets complexes (héritage, délégation…).

Ils rendent le système indépendant de la manière dont les objets sont créés, composés et représentés. Ils permettent dynamiquement ou statiquement de préciser Quoi (l'objet), Qui (l'acteur), Comment (la manière) et Quand (le moment) de la création.

Il existe deux types de motifs :

- Motifs de création de classe (utilisation de l'héritage) : Factory.
- Motifs de création d'objets (délégation de la construction à un autre objet) : AbstractFactory, Builder, Prototype.

Les patrons créateurs classés selon GoF sont :

- **La Fabrique Abstraite (AbstratFactory):** Elle fournit une interface pour créer des familles d'objets apparentés ou dépendants, sans avoir à spécifier leurs classes concrètes.
- **Le Monteur (Builder):** Il dissocie la construction de la représentation d'un objet complexe, de sorte que le même procédé de construction puisse engendrer des représentations différentes.
- **La Fabrication (Factory) :** Elle définit une interface pour la création d'un objet, tout en laissant à ses sous-classes le choix de la classe à instancier.
- **Le Prototype :** En utilisant une instance type, il crée de nouveaux objets par clonage.
- **Le Singleton :** Il garantit qu'une classe n'a qu'une seule instance, et fournit à celle-ci un point d'accès de type global.

I.2.9.2) Les patrons structurels : [1] [2]

Les patrons structurels proposent des schémas de classes et d'objets pour réaliser des structures plus complexes, ils permettent :

- Abstraction de la manière dont les classes et les objets sont composés pour former des structures plus importantes.
- Ajouter un niveau d'indirection pour accéder à un objet.

Ex : Adapter d'objet, Bridge, Façade, Proxy,

- Composition récursive pour organiser un nombre quelconque d'objets

Ex : Composite.

Il existe deux types de motifs :

- **Motifs de structure de classes :** Utilisation de l'héritage pour composer des interfaces et/ou des implémentations (ex : Adapter).
- **Motifs de structure d'objets :** composition d'objets pour réaliser de nouvelles fonctionnalités.

Les patrons structurels classés selon GoF sont :

- **L'adaptateur (Adapter) :** Il convertit l'interface d'une classe existante en une interface conforme à l'attente de l'utilisateur. Il permet à des classes de travailler ensemble malgré leur incompatibilité d'interface.
- **Le Pont(Bridge) :** Il découple une abstraction de son implémentation, afin que les deux puissent être modifiés indépendamment.
- **Le Composite :** Il organise les objets en structure arborescente représentant une hiérarchie de composition. Il permet un traitement uniforme des objets individuels, et des objets composés.
- **Le Décorateur (Decorator) :** Il attache des responsabilités supplémentaires à un objet de façon dynamique. Il offre une solution alternative à la dérivation de classes pour l'extension de fonctionnalités.
- **La Façade(Facade) :** Elle fournit une interface unifiée pour un ensemble d'interfaces d'un sous-système. Elle définit une interface de plus haut niveau, qui rend le sous système plus facile à utiliser.
- **Le Poids mouche(Flyweight) :** Il supporte de manière efficace un grand nombre d'instances de granularité fine.
- **La procuration (proxy) :** Elle permet de remplacer temporairement un objet par un autre, pour en contrôler l'accès.

I.2.9.3) Les patrons comportementaux : [1] [2]

Ils traitent du placement des algorithmes entre plusieurs classes et simplifient la dynamique des responsabilités entre les objets.

Les patrons comportementaux sont utilisés :

• Pour décrire comment des groupes d'objets coopèrent (ex : Mediator)

• Pour définir et maintenir des dépendances entre objets (ex : Observer)

• Pour encapsuler un comportement dans un objet et déléguer les requêtes à d'autres objets (Ex : Strategy, State, Command)

• Pour parcourir des structures en appliquant des comportements (ex : Visitor, Iterator).

Il existe deux types de motifs :

- Motifs de comportement de classes : utilisation de l'héritage pour répartir les comportements entre des classes (ex : Interpreter).
- Motifs de comportement d'objets avec l'utilisation de l'association entre objets :

Les patrons comportementaux classés selon Gof sont :

- **La Chaîne de responsabilité :** Elle permet de découpler l'expéditeur d'une requête à son destinataire, en donnant la possibilité à plusieurs objets de prendre en charge la requête.

De ce fait, une chaîne d'objets récepteurs est créée pour faire transiter la requête jusqu'à ce qu'un objet soit capable de la prendre en charge.

- **La Commande (Command) :** Elle réifie une requête, ce qui permet de faire un paramétrage des clients avec différentes requêtes, files d'attentes, ou historique de requêtes, et d'assurer le traitement des opérations réversibles.
- **L'Interprète(Interpreter) :** Pour un langage donné, il définit une représentation objet de la grammaire utilisable par un interprète pour analyser des phases du langage.
- **L'Itérateur(Iterator):** Il fournit un moyen pour accéder en séquence aux éléments d'un objet de type agrégat sans révéler sa représentation sous-jacente.
- **Le Médiateur(Mediator):** Il définit un objet qui encapsule les modalités d'interaction de divers objets. Il factorise les couplages faibles, en dispensant les objets d'avoir à faire référence explicite les uns aux autres ; de plus, il permet de modifier une relation indépendamment des autres.
- **Le Mémento :** Sans violer l'encapsulation, il acquiert et délivre à l'extérieur une information sur l'état interne d'un objet, afin que celui-ci puisse être rétabli ultérieurement dans cet état.
- **L'Observateur(Observer):** Il définit une corrélation entre des objets de façon que lorsqu'un objet change d'état, tous ceux qui en dépendent, en soient notifiés et mis à jour automatiquement.
- **L'Etat(State):** Il permet à un objet de modifier son comportement lorsque son état interne change.
- **La Stratégie(Strategy):** Elle définit une famille d'algorithmes, les encapsule, et les rend interchangeables. Une stratégie permet de modifier un algorithme indépendant de ses clients.
- **Le Patron de méthode (Template method) :** Il définit le squelette de l'algorithme d'une opération, en déléguant le traitement de certaines étapes à ses sous-classes. Le patron de méthode permet aux sous-classes de redéfinir certaines étapes d'un algorithme sans modifier la structure de l'algorithme.
- **Le Visiteur (Visitor):** Il représente une opération à effectuer sur les éléments d'une structure d'objet. Le visiteur permet de définir une nouvelle opération sans modifier les classes des éléments sur lesquels il opère.

Conclusion :

Au cours de ce chapitre nous avons dévoilé les patrons ainsi que ces différents types y compris les patrons de conception, nous avons également présenté quelques exemples de patrons

(classe, ancêtre, conteneur, utilisateur, messager), on a aussi défini le Framework et les techniques de modélisation. En plus nous avons défini les patrons de conception et on les a classés selon GoF (Créateur, structurel, comportemental).

Dans le prochain chapitre, nous allons découvrir l'intégration et l'interopérabilité.

CHAPITRE II

INTEGRATION ET INTEROPERABILITE

Introduction :

L'intégration des systèmes consiste à assembler les différentes parties d'un système et à assurer leur compatibilité ainsi que le bon fonctionnement du système complet. Il s'agit donc de faire tomber les barrières fonctionnelles et organisationnelles eu sein des entreprises afin que l'ensemble soit vu comme un tout cohérent. L'intégration système est une composante de l'ingénierie système, qui définit les méthodologies, méthodes, méta-modèles, modèles, langages et outils nécessaires à l'analyse et la conception d'un système.

L'intégration n'est pas un but en soi. C'est un moyen d'assurer la cohérence fonctionnelle et informationnelle de l'entreprise. L'interopérabilité des systèmes, telle que définie par la suite, n'est elle aussi qu'un moyen, parmi d'autres, pour faciliter l'intégration.

L'interopérabilité des systèmes n'est pas un problème nouveau. C'est principalement une mode, et nombre de problèmes d'interopérabilité ont déjà été traités par des nombreuses recherches sur l'intégration (Vernadat 1996 ; Bernus, et al. 1998).

Cependant, alors que l'intégration concerne aussi les aspects organisationnels de l'entreprise, l'interopérabilité se focalise principalement sur les aspects techniques. Elle concerne à la fois les systèmes matériels et les systèmes logiciels.

II.1. Intégration de système :

II.1.1.Qu'est-ce qu'un système? [10]

Ensemble composite constitué de personnels, de matériels, de logiciels, de procédures en interaction mutuelle dans un environnement donné organisés pour répondre à un besoin correspondant à une certaine finalité.

L'intégration du Système d'Information de Gestion (SIG) est depuis une dizaine d'années maintenant à la fois un enjeu et un défi majeurs pour la plupart des entreprises :

- **Un enjeu technologique et organisationnel** : En raison de l'hétérogénéité des acteurs, des processus, des données, des applications et des composants à faire fonctionner ensemble.
- **Un défi contextuel** : En raison des multiples questions posées par un agenda de plus en plus difficile à appréhender pour l'organisation devant répondre à un environnement toujours plus exigeant en termes de délais, de volumes et de pertinence.

II.1.2.Qu'est-ce que l'intégration ? [11]

L'intégration est une phase d'un projet durant laquelle on vérifie le produit par des tests d'intégration .

Le terme intégration désigne également la conception et la réalisation d'un système d'information intégré par la mise en relation (interfaçage) de différents logiciels ou matériels existants.

Ce défi de l'intégration est également perturbant pour l'organisation qui reste confrontée à un marché asymétrique ; où d'une part le discours de l'offre (éditeurs, consultants, intégrateurs, etc.) est à la fois rassurant et inquiétant ; et où d'autre part le mimétisme des demandeurs (clients, partenaires, etc.) ajoute à leur vulnérabilité.

L'intégration d'applications est un concept clé pour toute entreprise qui veut rester compétitive ou bénéficier d'un avantage concurrentiel. Mais devant la prolifération de solutions, de technologies et d'approches d'intégration, il devient de plus en plus difficile de maîtriser l'éventail des connaissances liées à ce domaine.

Au sein de l'entreprise, il peut exister plusieurs approches permettant d'appréhender le problème d'intégration. Principalement, on peut distinguer quatre types fondamentaux. Il s'agit respectivement en fonction de leur degré de complexité, de l'intégration de données, de processus, des interfaces et des applications.

II.1.2.1. Intégration de données : [11]

C'est la forme la plus simple de l'intégration. Elle apparaît au niveau des bases de données. D'une part, elle est assurée par duplication des copies d'une partie ou de toute la base de données dans une ou plusieurs applications. D'autre part, l'intégration s'effectue par le transfert des données, en utilisant des outils pour permettre aux données d'émigrer d'une application à une autre. Ce transfert de données est généralement réalisé par ETL (Extract Transform and Load). ETL est un moteur qui extrait, transforme, épure puis charge les données à partir de différentes applications vers des entrepôts de données. Il est aujourd'hui la solution la plus préconisée dans l'intégration des données.

II.1.2.2. Intégration des applications : [11]

L'intégration d'applications porte sur l'interconnexion d'applications hétérogènes, le plus souvent développées de façon indépendante et voire de façon incompatible. L'AI permet principalement de faire communiquer tout type d'applications (CRM - Customer Relationship Management, ERP -Entreprise Ressource Planning, SCM - Supply Chain Management, etc.), ce qui peut constituer des enjeux énormes notamment pour les grosses entreprises qui disposent d'une masse importante d'applicatifs. Sur le terrain, l'AI s'affiche par une multitude de produits commerciaux portant des logos assez variés tels que EAI ou ESB (Business Work de Tibco, Integrator de Mercator, e*Gate Integrator de SeeBeyond, Websphere d'IBM, Biztalk de Microsoft, Businessware de Vitria, Intégration Server de WebMethods, EntireX de SoftwareAG,

XMLBus d'Iona, Sonic ESB de Sonic Software, etc.) , et dont l'objectif est de permettre de rationaliser et fluidifier le système d'information afin de le rendre plus flexible et plus réactif.

II.1.2.3. Intégration des processus : [11]

C'est la forme la plus complexe de l'intégration. Elle sert à rendre valable une application dans le contexte d'une autre sans la dupliquer. Elle permet aussi de construire de nouveaux processus métier à base des applications et progiciels existants. Ceci crée de nouvelles opportunités pour l'entreprise à moindre coût. Les données circulant dans la nouvelle organisation sont accédées et maintenues selon une logique de métier (business logic) qui a des règles et une sécurité de données. Ces données ne sont plus simples mais des objets métier (BOD : Business Object Document, ex bon de commande) qui portent déjà un sens. Grâce à cette forme d'intégration, les nouveaux processus métier qui les manipulent sont créés. L'intégration du SI passe par l'intégration des briques le composant étant présent à un moment donné. Aujourd'hui, la brique élémentaire du SI est l'application. C'est donc tout logiquement que l'intégration du SI est considérée comme l'intégration des applications qui le composent. Néanmoins, il ne suffit pas de connecter les applications entre elles selon les besoins en information de telle ou telle application pour dire que l'on fait de l'intégration de SI. Il ne faut pas prendre l'application comme un élément stable et autonome qu'il faudrait connecter à d'autres éléments stables et autonomes. Il faut plutôt intégrer ce pourquoi les applications ont été conçues. Il faut donc revenir à leur processus de conception afin de définir l'objet de l'intégration de SI.

II.1.3.Intégration d'applications d'entreprise(EAI) : [11]

L'EAI (Enterprise Application Intégration) est un terme qui regroupe les méthodes et les outils visant à moderniser, consolider et coordonner les différentes solutions logicielles d'une entreprise. Typiquement, une entreprise dispose d'applications et de bases de données qu'elle désire continuer à utiliser tout en développant ou en migrant de nouvelles applications dans le but d'exploiter un site Web (e-Commerce) ou construire un extranet avec ses partenaires.

Elle peut également vouloir ajouter de nouvelles fonctionnalités à ses applicatifs, afin de les pérenniser, mais ne pas vouloir/pouvoir modifier ses systèmes d'information. Dans ce cas, elle peut développer une solution en intranet et, grâce à l'EAI, la relier aux autres systèmes existants. De plus, il est ainsi possible d'effectuer un reporting centralisé (consolidé) des données en provenance de sources très diverses. Ainsi, l'EAI peut être vu comme une meta_structure coordonnant des applicatifs intranet, Internet.

L'objectif majeur des technologies d'intégration est de rendre l'entreprise plus efficace, plus rentable et plus accessible aux tiers ; en d'autre termes, d'en faire un partenaire commercial plus efficace.

L'EAI sert aussi à réaliser les objectifs suivants :

- ✓ Créer une infrastructure intégrée pour relier des systèmes (applications et sources de données) dispersés au sein de l'entreprise incorporée.
- ✓ Fournir une solution full duplex et bidirectionnelle pour partager des informations de manière similaire entre les diverses applications de l'entreprise et les nouveaux progiciels commerciaux.
- ✓ Permettre l'échange de données et de processus entre les différentes applications d'une entreprise de manière rapide, efficace et transparente.

II.1.3.1.Avantages de l'EAI: [11]

- **Flexibilité** : Une modification dans une application n'a d'impact que sur le serveur d'applications, et non sur les X destinataires qui l'utilisent,
- **Robustesse** : La centralisation des flux permet un réel suivi, des sauvegardes, des reprises, etc.,
- **Sécurité** : Les technologies d'intégration se fondent sur des mécanismes asynchrones et peuvent offrir une gestion poussée de la sécurité. L'EAI offre une infrastructure qui permet d'assurer la sécurité des flux de données véhiculés et les fonctions d'administration et d'exploitation.
- **Modularité** : L'EAI donne la possibilité de modéliser les processus et de rationaliser les échanges inter-applicatifs au sein du SI.
- **Gestion de l'hétérogénéité** : L'EAI permet d'encapsuler l'hétérogénéité par unification des interfaces et alignement des processus métiers.
- **Flux centralisés** : Avant l'arrivée de l'EAI, les entreprises devaient développer des interfaces spécifiques à chaque application et les connecter point à point. Il en résultait un réseau complexe (plat de spaghetti) de flux, difficile à maintenir et à faire évoluer. Maintenant, toutes les interfaces EAI convergent vers un serveur central (concentrateur ; en anglais, *hub*) qui traite et redistribue les flux vers les applications enregistrées.
- **Flux traités « au fil de l'eau »** : Les mises à jour des données sont effectuées au fil de l'eau, c'est-à-dire au fur et à mesure des événements des applications sources. Cela réduit les flots de données lors des transferts et propose une donnée "à jour" peu de temps après son éventuelle modification. Cela réduit aussi la perte de performance des applications

due à l'extraction ou la mise à jour des données car on ne traite que des flots de petite taille et répartis dans le temps.

- **Flux réutilisable** : Si une nouvelle application veut accéder aux OM (Objets de Métier) déjà présents dans l'IAE, toute la logique de récupération n'est plus à développer. En théorie elle n'a besoin d'ajouter au concentrateur IAE que sa *collaboration* (si elle a besoin d'un traitement spécifique), ses *OMS* (Objets de Métier Spécifiques), ses *mappings* (la mise en cohérence entre deux types d'informations distincts) et son *connecteur*.
- **Coût de migration des interfaces** : Lors du changement d'une des applications interfacées (migration, changement de produit), peu de modifications sont nécessaires. Seuls le connecteur, le mappage ou la collaboration spécifique à l'application doivent être modifiés.

II.1.3.2.Inconvénients de l'EAI : [11]

- **Flux massif** : Pour les flux massifs (par exemple : mise à jour de 10 000 articles en même temps), la logique du traitement unitaire de l'information est très lente. On préfèrera plutôt une solution ETL ((*Extraction, Transformation, Loading)*, est un système de chargement de données depuis les différentes sources d'information de l'entreprise jusqu'à l'entrepôt de données).
- **Coût initial** : Le coût de mise en place de l'infrastructure est assez élevé. Mais il se réduit grandement au fur et à mesure de l'ajout de nouveaux flux.
- **Resynchronisation des bases** : A la suite d'un incident (bug applicatif, erreur d'exploitation, endommagement de disque, ...), ou encore de l'enrichissement des structures de données, il faut resynchroniser les bases où les données sont copiées avec celle où les données sont en référence. Ce phénomène est malheureusement quasi certain, et même assez fréquent. Une procédure spéciale de resynchronisation est généralement nécessaire. Elle travaille sur des données statiques pouvant être volumineuses et non plus sur des événements. Une étude fonctionnelle est impérative. Il faut souvent ajouter des données de resynchronisation dans les bases (identifiant de rapprochement, date-heure de dernière mise à jour, ...). De fait, il faut doubler l'EAI de fonctionnalités plus proches de celles d'un ETL. Avant de proposer une mise à jour au fil de l'eau, il convient de commencer par étudier la procédure de resynchronisation. Il est fréquent qu'elle suffise à répondre au besoin. Sinon le génie de l'architecte doit s'exprimer pour trouver une solution modulaire et éviter la redondance des règles métier entre les deux outils. Il est souvent suffisant de répliquer par exemple toutes les 4h les données modifiées les 5

derniers jours. La période de cinq jours limite le volume. Le recouvrement de période permet, lorsque la procédure tombe en erreur ou lorsque la base source est restaurée ou réparée, de rétablir automatiquement la cohérence lors du passage suivant. La même procédure est utilisée pour réaligner l'intégralité des données en jouant sur la période. Ainsi, les règles de réplication sont toutes dans le même outil.

II.1.4. Principe de fonctionnement de L'EAI : [11]

Une plate-forme EAI fonctionne sur le modèle d'une multiprise. Chaque application possède un connecteur standard (la prise) qui est relié au « bus EAI » (la multiprise). Le connecteur est un exécutable ou une classe Java, installé sur la machine qui héberge l'application. Il traduit les données provenant de l'application dans un format lisible par un courtier de message, et vice versa. Il existe deux types de connecteurs : techniques et applicatifs:

- Les connecteurs techniques sont reliés aux applications depuis leur base de données, des fichiers plats, etc.
- Les connecteurs applicatifs interfacent directement leurs API (Application Programing Interface).

Encore propriétaire au début des années deux mille, les connecteurs se standardisent peu à peu autour de technologies telles que les web services (WSDL, Soap, HTTP) ou JCA (J2EE Connector Architecture). Au cœur de la plate-forme, le « bus EAI » traditionnel est constitué d'un courtier de messages (message broker) et d'un MOM (middleware orienté messages). Le courtier de messages applique des transformations sur les messages entrants avant de les renvoyer vers les applications. Il est également capable de router une information sur une file d'attente particulière du MOM. Ainsi, si une application destinataire n'est pas accessible, le MOM stocke les messages entrants et sortants jusqu'à ce qu'ils soient récupérés par leurs destinataires. C'est ce mécanisme de communication asynchrone qui permet de découpler les applications les unes des autres. Dans cette architecture de type « publication et abonnement », chaque application s'abonne à des files de messages sur lesquelles elle peut publier et recevoir des messages, le plus souvent aujourd'hui au format XML. Il existe un autre modèle qui consiste à relier directement deux applications entre elles via un mécanisme de type RPC (Remote Procedure Call ou appel de procédure distante).

II.1.5.Services de l'EAI : [11]

Les fonctionnalités essentielles d'une solution EAI sont regroupées en services:

- **Le service d'interfaçage :** Est assuré par les connecteurs, qui se chargent d'intégrer les API vers un transport standard (fichier, émail,…) vers un progiciel (ERP, CRM,…), et de

réaliser la correspondance entre la structure des informations en provenance ou à destination de l'extérieur.

- **Le service de queuing :** Permet de garantir l'intégrité de l'information tout au long du traitement du flux dans la plate-forme d'intégration. Il remplit le rôle de gestionnaire de messages, réceptionne les évènements produits par les connecteurs et les propage aux composant d'intégrations internes ou aux connecteurs abonnés publier et s'inscrire.
- **Le service de pilotage :** Envoyer des commandes de pilotage (arrêt, démarrage,…) et de réceptionner les alertes émises par les composants d'intégrations.

II.1.6.Types de projets d'intégration : [11]

L'EAI intéresse toutes les entreprises qui ont une implantation multi-sites, des matériels différents, des bases de données multiples, des systèmes multiapplicatifs ou des problématiques B2B. Toute entreprise a besoin d'un EAI à des degrés divers en fonction de sa configuration et de ses systèmes, comme en fonction de ses moyens.

Un projet d'intégration est généralement un projet complexe et d'envergure dans la mesure où il est rarement ponctuel, et de plus il concerne et impacte l'ensemble du SI. Pour cette raison, il doit s'inscrire dans une optique de projet global d'infrastructure du SI. Ce projet peut être réalisé selon une démarche descendante (top-down) et globalisante dans la mesure où elle touche à la globalité de l'entreprise, et on parle généralement de projets stratégiques ou d'infrastructure. Toutefois, il demeure possible d'adopter une démarche ascendante (bottomup) conduisant ainsi à des projets tactiques.

II.1.6.1. Projets EAI stratégiques : [11]

L'aspect stratégique concerne l'intégration des SI de l'entreprise ainsi que ses partenaires B2B. L'intégration stratégique demande beaucoup plus de moyens et de temps, qui n'est pas forcément couronné du succès escompté. L'EAI stratégique implique une prise en compte et une évolution éventuelle du SI global de l'entreprise et de ses process. Ce sont des solutions qui se justifient dans des environnements applicatifs complexes ou des environnements critiques, de grandes entreprises. L'EAI devient alors l'un des piliers de l'architecture du SI.

II.1.6. 2. Projets EAI tactiques : [11]

Le terme tactique signifie un périmètre limité et parfaitement ciblé. L'intégration tactique nécessite souvent des moyens faibles en ressources, temps et budget, et dispose un retour sur investissement rapide, cependant elle doit être menée comme un morceau du puzzle d'intégration stratégique, puzzle que l'entreprise sera peut être amenée à faire dans l'avenir.

II.2. L'interopérabilité :

II.2.1.Définition de l'interopérabilité : [12]

L'interopérabilité est la capacité que possède un produit ou un système dont les interfaces sont intégralement connues à fonctionner avec d'autres produits ou systèmes existants ou futurs. Il convient de distinguer interopérabilité et compatibilité. Pour être simple, on peut dire qu'il y a compatibilité quand deux produits ou systèmes peuvent fonctionner ensemble et interopérabilité quand on sait pourquoi et comment ils peuvent fonctionner ensemble. Autrement dit, on ne peut parler d'interopérabilité d'un produit ou d'un système que si on en connaît intégralement toutes ses interfaces.

L'interopérabilité est considérée comme très importante voire critique dans de nombreux domaines, dont l'informatique, le médical au sens large, les activités ferroviaires, l'électrotechnique, l'aérospatiale, le domaine militaire et l'industrie en général. Les différents systèmes, appareils et éléments divers utilisés doivent pouvoir interagir sans heurts.

Pour définir plus exactement ce qu'est et n'est pas l'interopérabilité, on peut commencer par la distinguer de la compatibilité. Cette dernière relation est binaire et concerne un ensemble fini de systèmes. A et B sont compatibles, ou pas, si leurs constructions respectives leur permettent, ou pas, de communiquer et travailler ensemble.

A et B seront dit interopérables si, grâce à une ou plusieurs norme(s) externe(s) qu'ils respectent, ils en viennent entre autre à pouvoir être compatibles. L'interopérabilité est générale et ne concerne pas à priori des éléments ou systèmes particuliers. Elle existe à travers des normes et formats respectés par tout élément ou système qui souhaite intégrer un plexus interopérable, le réseau des éléments qui communiquent entre eux de façon fluide et normée. On voit que l'interopérabilité ne doit rien au hasard, et résulte d'un accord explicite entre les différents constructeurs d'éléments.

II.2.2.Interopérabilité en informatique : [13]

L'interopérabilité ou interfonctionnement en informatique est la capacité que possède un système informatique à fonctionner avec d'autres produits ou systèmes informatiques, existants ou futurs, sans restriction d'accès ou de mise en œuvre.

Cette capacité facilite au citoyen d'utiliser l'informatique sans se soucier des aspects techniques. Elle permet ainsi , sans préjudice, d'obliger par les ordres, qu'il donne à un ordinateur, par l'intermédiaire d'un programme, de toute nature, de se coordonner, de coopérer et d'être piloté par tout autre programme d'une autre nature quel que soit le lieu, le matériel et le langage utilisé. Il doit le faire si le service attendu l'exige. Le service rendu doit être de même

valeur satisfaisante qu'il eut été fait par l'un ou l'autre des programmes, dès lors que le service correspond, sans obstacle contractuel ni obstacle technique.

II.2.3. Interopérabilité partielle : [14]

Malgré ces définitions, en pratique, l'interopérabilité n'est pas par elle-même un élément concret ou un critère défini. On peut déterminer dans quelle mesure des systèmes sont inter opérables en jugeant de leur respect de la norme qui a donné lieu à une interopérabilité. Ce qui nous amène à la notion d'interopérabilité partielle : si un logiciel, par exemple, ne respecte qu'une partie d'une norme, il ne pourra peut-être pas dialoguer correctement avec un autre programme, voire pas du tout. Dans l'absolu, seul le respect strict d'une norme donnée conduit à une interopérabilité réelle, mais cette situation est assez éloignée de la réalité.

II.2.4.Définition de compatibilité : [14]

La compatibilité est stipulée dans le même document et permet de bien faire la différence entre ces deux notions différentes mais complémentaires : *« On entend par compatibilité la capacité de deux systèmes à communiquer sans ambiguïté ».*

Deux systèmes peuvent donc être compatibles mais non inter opérables. On ne peut parler d'interopérabilité que lorsque des systèmes quelconques (par exemple des logiciels, des systèmes de gestion de base de données, des périphériques, etc… produits par des entreprises différentes, fonctionnant sur des systèmes d'exploitation différents) peuvent communiquer entre eux.

II.2.5. Les principes d'interopérabilité : [13] [15]

L'interopérabilité se base sur des principes :

- **Complexité de l'interopérabilité en informatique :**

L'informatique pose le problème de l'interopérabilité en des termes nouveaux. Elle met en évidence certaines contradictions entre les intérêts commerciaux d'entreprises fournissant produits et services, et les exigences nouvelles des consommateurs de ces produits et services. Du fait des outils informatisés, de l'expertise acquise par des groupes d'utilisateurs, de la communication facilitée, l'interopérabilité devient une problématique plus concrète aux yeux d'un nombre grandissant de personnes, qui en comprennent mieux les tenants et aboutissants notamment les enjeux du choix et de la protection des données.

Ce mouvement est vu comme une avancée démocratique par les partisans d'une interopérabilité « ouverte », mais cet avis n'est pas partagé par tous. Nombres d'entreprises défendent à l'inverse un modèle plus classique où l'interopérabilité reste le fruit de l'initiative privée et subit un contrôle strict. De par les enjeux qui lui sont aujourd'hui liés, dans les domaines du travail ou dans la sphère privée par exemple, l'interopérabilité informatique va certainement jouer un rôle de catalyseur des changements futurs, quels qu'ils soient.

- **Nécessité des normes :**

L'interopérabilité nécessite que les communications obéissent à des normes, clairement établies et univoques. Ces documents techniques définissent souvent des exigences, parfois accompagnées de recommandations plus ou moins optionnelles. Si la norme est correctement écrite, deux systèmes qui satisfont aux exigences doivent dialoguer ensemble sans souci particulier. Ils peuvent ainsi évoluer librement sans risque de casser cette possibilité de communication, tant qu'ils respectent la norme définissant leurs interfaces.

- **Les données véhiculées dans les interfaces :**

En pratique, l'interopérabilité touche tous les domaines de l'informatique. Ce sont les règles de cohérence des données véhiculées qui gouvernent l'interopérabilité. Ainsi, les données de référence employées par plusieurs applications sont généralement celles qui pilotent l'interopérabilité. Dans des contextes où coexistent les données structurées (les bases de données) et les données non structurées (les documents, textes, images), on considère généralement aujourd'hui que les données communes sont constituées par les métadonnées.

Par conséquent, les interfaces de programmation (API) sont à la base de l'interopérabilité informatique, ces API peuvent s'appliquer à différents types de ressources informatiques (bases de données) ou application.

- **Interfaces de programmation :**

Les interfaces de programmation (API) sont à la base de l'interopérabilité informatique. Par exemple, la spécification J2EE pour le langage de programmation Java comporte de nombreux types d'API, qui véhiculent des métadonnées. Ces API peuvent s'appliquer à différents types de ressources informatiques (bases de données) ou applications (Progiciel de gestion intégré).

II.2.6.Les niveaux d'interopérabilité :

On distingue 3 niveaux d'interopérabilité :

II.2.6.1. L'interopérabilité technique : [15]

L'interopérabilité technique désigne le recours à la définition et l'utilisation d'interfaces technologiques, des normes et des protocoles, en vue de créer des systèmes d'information collaboratifs fiables, efficaces et performants capables d'échanger l'information.

Sur un plan technique, l'interopérabilité se réalise à trois niveaux techniques complémentaires :

1. Une description des ressources avec sémantiques communes issues de différents jeux de métadonnées standardisées.
2. Un contexte générique d'implémentation de ces descriptions dans des langages structurés standardisés, interprétables par les machines.

3. Des protocoles informatiques d'échange de ces données normalisées.

Le tableau suivant résume le positionnement les niveaux d'interopérabilité technique vu précédemment sous des différents standards traditionnels et récents :

	Standards traditionnels	Standards récents
Jeux de métadonnées	MARC	Dublin core MARC-XML, MODS , EAD LOM …
Cadre générique D'implémentation	ISO 2709 ISAD(G)	XML RDF Espace de nom URL
Protocoles	WAIS FTP Z39.50	HTTP OAI-PMH SRU/SRW

Figure II.1 : Niveaux d'interopérabilité technique

II.2.6.2. L'interopérabilité sémantique: [15]

L'interopérabilité sémantique consiste à faire en sorte que la signification des informations échangées ne soit pas perdue dans la procédure d'interopérabilité et qu'elle soit préservée et comprise par les personnes, les applications et les institutions concernées (les formats des messages, la structuration, la sémantique des constituants). Elle nécessite que l'interopérabilité technique soit effective.

II.2.6.2.1.Les principes d'interopérabilité sémantique : [15]

L'interopérabilité sémantique se base sur quatre principes nécessaires qui sont :

a. : Métadonnées

Les métadonnées se sont des informations descriptives sur les ressources. L'utilisation de métadonnées descriptives et standardisées améliore la recherche des informations pertinentes dans un réseau de ressources.

b. Ontologie :

L'ontologie est une description formelle des concepts, des rôles et des relations qui existent pour un agent ou une communauté d'agents. Elles fournissent une compréhension commune d'un domaine qui peut être communiquée, tout en jouant un rôle majeur dans les échanges d'information.

c. Médiateurs :

Un médiateur est un adaptateur de données situées sur un réseau entre un client et un serveur de données (le client peut être une autre base de données). Par ailleurs, un médiateur est un composant logiciel qui résout les conflits schématiques et sémantiques.

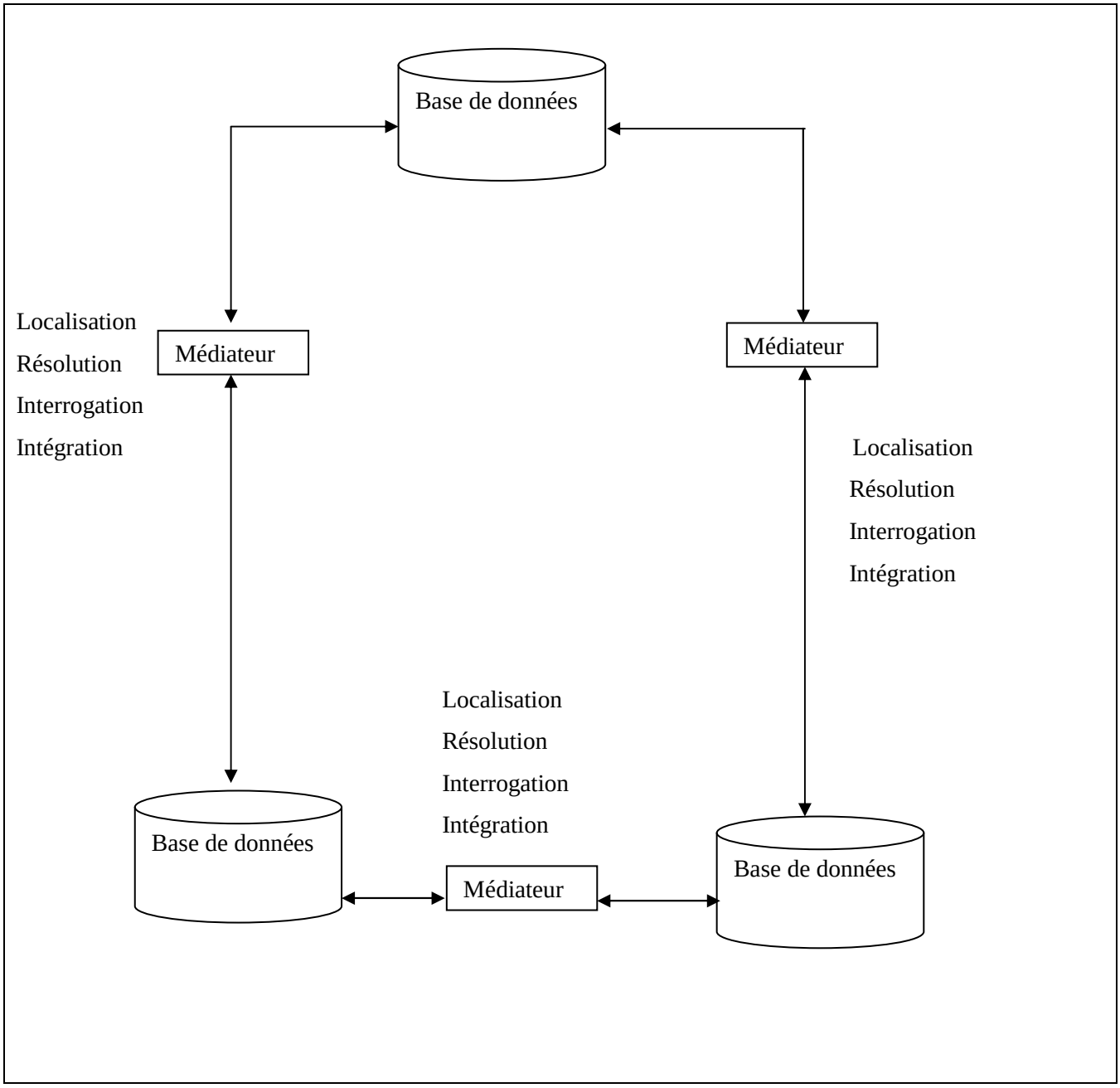

Figure II.2 : Principes d'interopérabilité sémantique

II.2.6.3.L'interopérabilité organisationnelle : [15]

L'interopérabilité organisationnelle est la capacité d'identifier les acteurs et les procédures organisationnelles intervenant dans la fourniture d'un service spécifique et de parvenir à un accord entre ces acteurs et à des procédures sur la manière de structurer leur interaction. En d'autres termes, il s'agit de définir les « interfaces d'entreprises » et de s'intéresser aux rôles des entités et des acteurs en interaction avec les systèmes d'information. Elle nécessite que l'interopérabilité sémantique soit effective.

II.2.6.3.1. Les niveaux d'interopérabilité organisationnelle : [15]

L'analyse des différents systèmes dans une entreprise génère des besoins pour alimenter la recherche sur l'interopérabilité. En effet, l'interopérabilité se trouve aujourd'hui contrainte à des barrières dues à l'incompatibilité des différents niveaux d'une entreprise. Pour atténuer ces barrières, les travaux de recherches visent à fournir des solutions génériques. De point de vue entreprise, l'interopérabilité passe par trois étapes : identifier le besoin en interopérabilité, identifier les obstacles qui empêchent de l'accomplir et enfin développer les approches fondamentales pour atténuer et envelopper ces barrières.

Le besoin en interopérabilité est présenté sur quatre niveaux : données, services, processus et métier. Ce besoin est exprimé tant en intra-entreprise que pour le inter-entreprise.

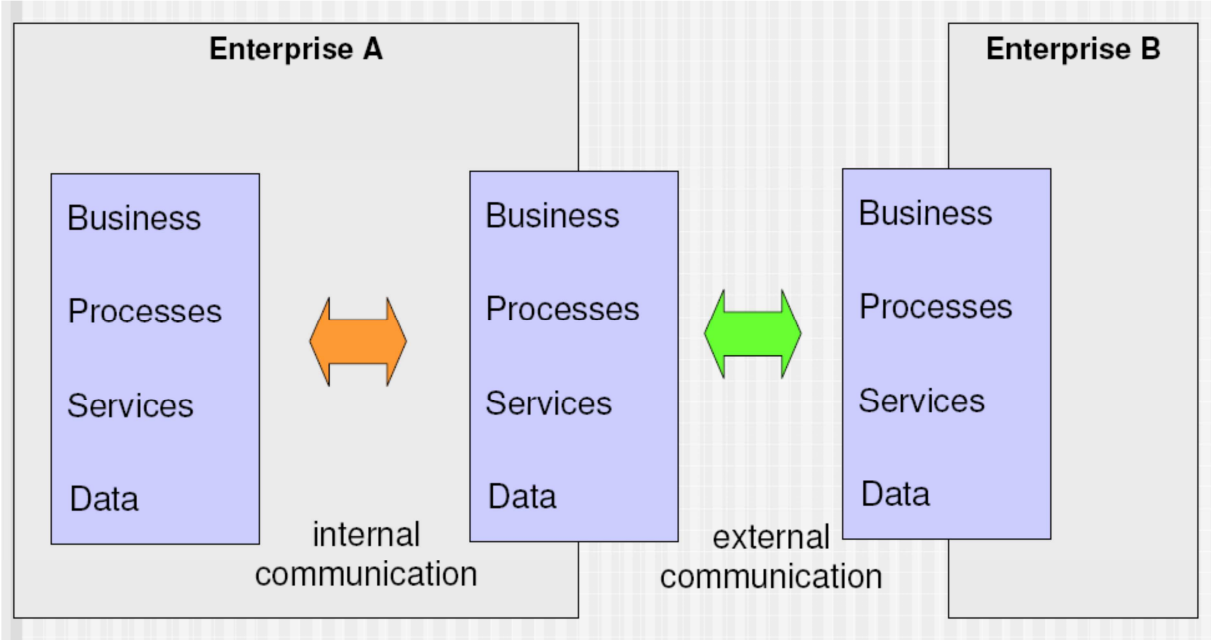

Figure II.3 : Niveaux d'interopérabilité organisationnelle

a) Interopérabilité au niveau des données

Du point de vue des données, l'objectif est de faire communiquer des modèles de données différents (hiérarchiques, Relationnel, etc.) ainsi que des langages de requêtes. En effet, les données sont organisées selon des schémas conceptuels différents (vocabulaires, structures de données, etc.) étroitement liés aux applications qui les supportent. L'interopérabilité des données revient donc à localiser et partager des informations provenant de sources hétérogènes appartenant à des bases de données différentes opérant sur des systèmes d'exploitation différents supportés par des machines différentes.

b) Interopérabilité au niveau des services

Il s'agit d'identifier, composer et rassembler des fonctions de différentes applications conçues et implémentées séparément. Ceci passe par la résolution des différences syntaxique et sémantique aussi bien que la connexion aux différentes sources d'information. Le terme « service » n'est pas limité à la notion de « web service » ou une application particulière, mais s'étend pour couvrir les fonctions d'une compagnie ainsi que les entreprises en réseaux.

c) Interopérabilité au niveau des processus

Un processus est définit par une séquence de services (fonctions) pour répondre à un besoin spécifique de l'entreprise. Généralement, ces processus évoluent en interaction (en série ou en parallèle). Dans un contexte inter entreprise, il faut étudier comment connecter des processus internes et créer de nouveaux processus en commun. L'interopérabilité inclut des mécanismes pour lier les langages de description des processus (les standards de workflow), des processus distribués et décentralisés ainsi que leur formation et vérification.

d) Interopérabilité au niveau des métiers (affaires)

Il s'agit d'acquérir la capacité à connecter, tant en interne à l'entreprise qu'en externe avec ses partenaires, les différentes spécifications métiers. Cette connexion doit se faire indépendamment de la vision interne d'une entreprise, de ses modèles métiers, de ses modes de décisions et de ses bonnes pratiques. Ceci facilite le développement et le partage des spécifications métiers entre les compagnies.

Ces différents niveaux de l'interopérabilité sont confrontés à trois types de barrières : des barrières d'ordre conceptuel provenant de la diversité des modes de présentation et de communication des concepts ; d'autres d'ordre technologique provenant de l'utilisation de technologies différentes pour communiquer et échanger des informations ; et finalement ceux d'ordre organisationnel provenant des différents modes de travail par exemple.

II.2.7. Interopérabilité et échange de données informatisées(EDI) : [14]

Interopérabilité et EDI (Electronic Data Interchange) sont deux concepts complémentaires. D'une part, si on introduit l'interopérabilité dans l'EDI, on obtient un échange de données ouvert à tout utilisateur potentiel, sans aucune restriction. D'autre part, l'EDI est le moyen qui permet la mise en place d'un système inter opérable, où des applications différentes, situées sur des ordinateurs différents, peuvent communiquer.

Il peut être judicieux aujourd'hui de concevoir un EDI inter opérable. En effet, l'administration française met en place actuellement un Référentiel Général d'Interopérabilité (RGI) dont l'objectif est « de fixer les règles techniques permettant d'assurer l'interopérabilité de tout ensemble de moyens destinés à élaborer, traiter, stocker, ou transmettre des informations faisant l'objet d'échanges par voie électronique entre autorités administratives et usagers ainsi qu'entre autorités administratives. » A terme, le RGI devrait donc permettre des échanges facilités entre l'administration et les entreprises. Les entreprises devront donc s'adapter à ce dispositif.

L'interopérabilité pourrait ainsi devenir une caractéristique indispensable des échanges de données à l'avenir.

A l'échelle internationale, une définition de l'interopérabilité a été donnée par l'IEEE (Institute of Electrical and Electronics Engineers) :

« La capacité, de deux ou plusieurs systèmes ou composants, à échanger de l'information et à utiliser l'information échangée. ». Même si cette définition est moins claire (pas de réelle différenciation des notions d'interopérabilité et de compatibilité), on voit que l'interopérabilité prend une dimension internationale. L'IEEE étant l'organisme représentant le plus d'acteurs du monde des TIC (plus de 360000 membres dans 175 pays), il est clair que l'interopérabilité devient un des enjeux majeurs de l'échange de données informatisé.

II.2.8. Fonctionnement de l'EDI : [14]

Le schéma suivant illustre le fonctionnement de l'EDI entre deux entreprises E1 et E2. Les quatre étapes décrites ci-dessous sont les fondements d'un EDI :

1 - La traduction d'un fichier du format propre à E1 au format standard d'échange par un traducteur spécifique à E1 ;

2 - L'envoi de ce fichier standardisé via un réseau fonctionnant sur un protocole de communication opté par E1 et E2 ;

3 - La réception du fichier de E1 standardisé par E2 ;

4 - La traduction du fichier standardisé au format propre à E2, par un traducteur spécifique à E2.

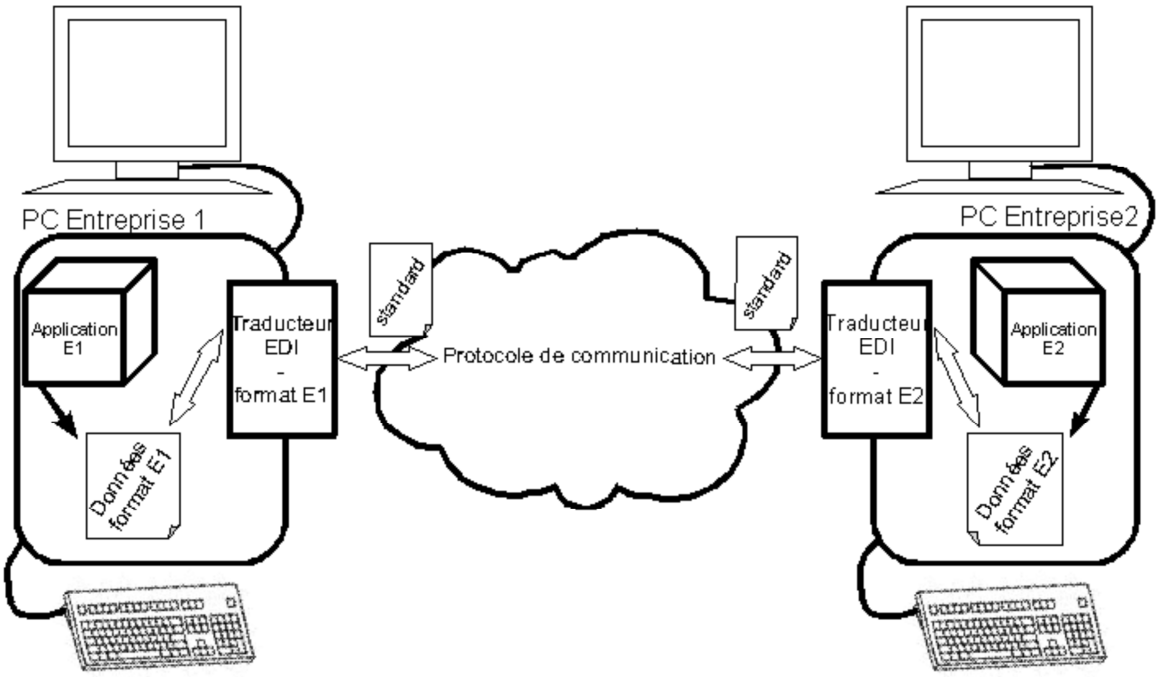

Figure II.4 : Fonctionnement de l'EDI

Le concept d'EDI reste donc relativement simple. La bonne compréhension de la terminologie du domaine de l'EDI permet d'éviter les confusions qui pourraient nuire aux projets. Cependant, l'EDI en application n'est pas aussi simple. En effet, l'évolution des technologies a conduit à l'apparition de variantes de l'EDI dont le principe de fonctionnement reste le même. Ces variantes ont pris le nom d'EEP (Echanges Electroniques Professionnels).

Conclusion :

A travers ce chapitre, nous avons présenté les concepts généraux d'intégration y compris ses principes, ses approches, ainsi que ses types de projets. De plus nous avons défini les concepts généraux d'interopérabilité y compris ses principes, ainsi que ses différents niveaux, tel que l'interopérabilité technique, l'interopérabilité sémantique et ses principes ainsi que l'interopérabilité organisationnelle et ses différents niveaux.

Dans le troisième chapitre on s'intéressera à l'analyse et la conception de notre cas d'étude.

Chapitre III

Analyse et conception

Introduction :

Au fil des années, les environnements informatiques sont devenus plus complexes et plus hétérogènes en raison de la diversité des besoins des clients et des innovations grandissantes de l'industrie informatique. L'intégration des applications et des processus métiers dans les entreprises est une question critique du point de vue de l'augmentation de l'efficacité et de la réduction des coûts.

Les objectifs sont alors entre autres d'améliorer la réutilisation des services, des temps de développement réduits et une meilleure maitrise des coûts de maintenance.

Cependant, la mise en place de l'intégration des différents systèmes est difficile à la fois aux niveaux conceptuels et techniques.

Pour aboutir à notre objectif de faire intégrer les différents sous systèmes d'une organisation afin d'avoir un seul système pour que l'ensemble soit vu comme un tout cohérent, nous avons comme modèle de solution les motifs de conception déjà vus en premier chapitre.

Nous avons alors dégagé deux axes de réflexion : sans respect du modèle MVC (Modèle-Vue-Contrôleur) et avec respect de ce modèle.

III.1. Le modèle MVC (Model-View-Controler) : [16]

Le design pattern Modèle-Vue-Contrôleur (MVC) est un pattern architectural qui sépare les données (le modèle), l'interface homme-machine (la vue) et la logique de contrôle (le contrôleur).

Ce modèle de conception impose donc une séparation en 3 couches :

- Le modèle : Il représente les données de l'application. Il définit aussi l'interaction avec la base de données et le traitement de ces données.
- La vue : Elle représente l'interface utilisateur, ce avec quoi il interagit. Elle n'effectue aucun traitement, elle se contente simplement d'afficher les données que lui fournit le modèle. Il peut tout à fait y avoir plusieurs vues qui présentent les données d'un même modèle.
- Le contrôleur : Il gère l'interface entre le modèle et le client. Il va interpréter la requête de ce dernier pour lui envoyer la vue correspondante. Il effectue la synchronisation entre le modèle et les vues.

  Le contrôleur doit donc :

1. Analyser la requête, soit extraire les informations dans l'URL.
2. Faire une mise à jour du modèle si nécessaire, en lui passant des paramètres.
3. Déterminer la vue à afficher et demander son affichage.

La synchronisation entre la vue et le modèle se passe avec le pattern Observer. Il permet de générer des événements lors d'une modification du modèle et d'indiquer à la vue qu'il faut se mettre à jour.

Voici un schéma des interactions entre les différentes couches :

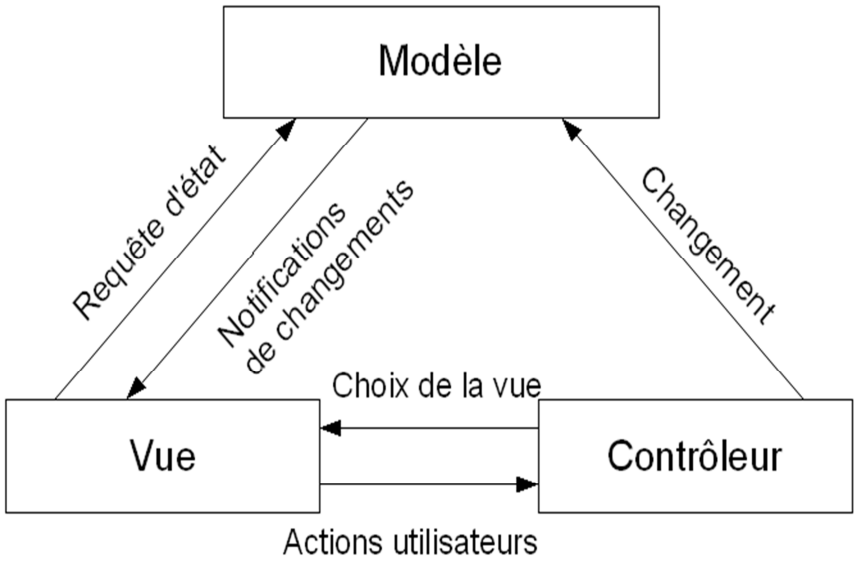

Figure III.1 : Modèle MVC

III.1.1. Développement sans respect du modèle MVC : [3]

Dans la plus part des architectures normalisées pour structurer une application, si une vue modifie les données, toutes les vues concernées par la modification doivent être mises à jour, c'est donc d'établir des règles du type « mettre à jour les vues concernant X si Y ou Z sont modifiés ». Mais ces règles deviennent rapidement trop nombreuses et ingérables si les relations logiques sont trop élevées. D'où :

- Réutilisation réduite.
- Temps de développement augmente.
- Coût de maintenance augmente.
- Minimiser la possibilité d'extension.
- Moindre lisibilité.

III.1.2. Développement avec respect du modèle MVC : [3]

Un avantage apporté par ce modèle est la clarté de l'architecture qu'il impose. Cela simplifie la tâche du développeur qui tenterait d'effectuer une maintenance ou une amélioration sur le projet.

En effet, la modification des traitements ne change en rien la vue. Par exemple on peut passer d'une base de données de type SQL à XML en changeant simplement les traitements d'interaction avec la base, et les vues ne s'en trouvent pas affectées. D'où :

- Possibilité de réutilisation.
- Temps de développement diminue.
- Facilité de maintenance.
- La possibilité d'extension.
- Plus de lisibilité.

III.1.2.1 Avantages de MVC : [17]

- Il peut y avoir plusieurs vues sur le même modèle
- Plusieurs contrôleurs peuvent modifier le même modèle
- Toutes les vues seront notifiées des modifications

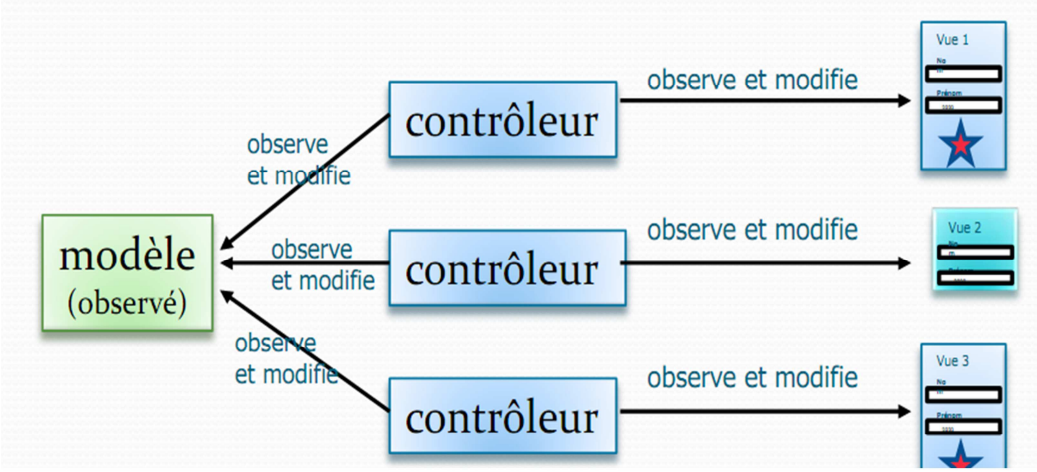

Figure III.2 : Fonctionnement de MVC

Notre but principal est de proposer une solution générale aux problèmes d'utilisateurs manipulant des données volumineuses et complexes d'où le but de ce modèle.

Alors on l'appliquera dans notre cas d'étude en dégageant les patrons de conception utilisés pour le faire.

III.2. Présentation de cas d'étude :

Considérons un système à développer qui est constitué de trois bases de données hospitalières **base1**, **base2** et **base3** des trois villes : **ville1**, **ville2** et **ville3** situées dans une même région dans cet ordre.

La base1 : est définie selon tables suivantes :

Hôpital (num_hop, nom_hop, adr_hop, propriété_hop);

Service (num_ser, num_hop, nom_ser, bât_ser, directeur_ser) ;

Salle (num_sal, num_ser, nbLits, étage) ;

Le numéro de salle est local à un service, i.e., il peut y avoir des salles avec le même numéro dans des services différents d'un même hôpital.

Employé (num_emp, nom_emp, adr_emp, tél_emp, situation_familiale_emp) ;

Docteur (num_doc, spécialité_doc, grade_doc) ;

Infirmier (num_inf, salaire_inf) ;

Un employé est soit infirmier soit docteur (num_inf et num_doc font référence à num_emp).

La base2 : est définie selon les tables suivantes:

Hospice (num_hos, nom_hos, adr_hos, propriété_hos) ;

Subdivision (num_subd, num_hos, nom_subd, bât_subd, directeur_subd) ;

Chambre (num_cham, num_subd, nbLits_cham, étage_cham) ;

Le numéro de chambre est local à une subdivision, i.e., il peut y avoir des chambres avec le même numéro dans des subdivisions différentes d'un même hospice.

Fonctionnaire (num_fonc, nom_fonc, adr_fonc, tél_fonc, situation_familiale_fonc) ;

Médecin (num_med, spécialité_med, grade_med) ;

Aide_soignant (num_aids, salaire_aids) ;

Un fonctionnaire est soit aide soignant soit médecin (num_aids et num_med font référence à num_fonc).

La base3 : est définie selon les tables suivantes :

Sanatorium (num_san, nom_san, adr_san, propriété_san);

Compartiment (num_comp, num_san, nom_comp, bât_comp, directeur_comp);

Pièce (num_piece, num_comp, nbLits_piece, étage_piece);

Le numéro de pièce est local à un compartiment, i.e., il peut y avoir des pièces avec le même numéro dans des compartiments différents d'un même sanatorium.

Travailleur (num_trav, nom_trav, adr_trav, tél_trav, situation_familiale_trav) ;

Soignant (num_soig, spécialité_soig, grade_soig) ;

Infirmier (num_inf, salaire_inf) ;

Un travailleur est soit infirmier soit soignant (num_inf et num_soig font référence à num_trav).

III.3. Description de système actuel :

Comme le montre la figure III.3 notre système contient trois bases de données (Base1, Base2, Base3) qui se sont reliées entre eux par un réseau.

A chaque fois que l'utilisateur a besoin d'une donnée, il doit accéder à toutes les bases pour qu'il la trouve.

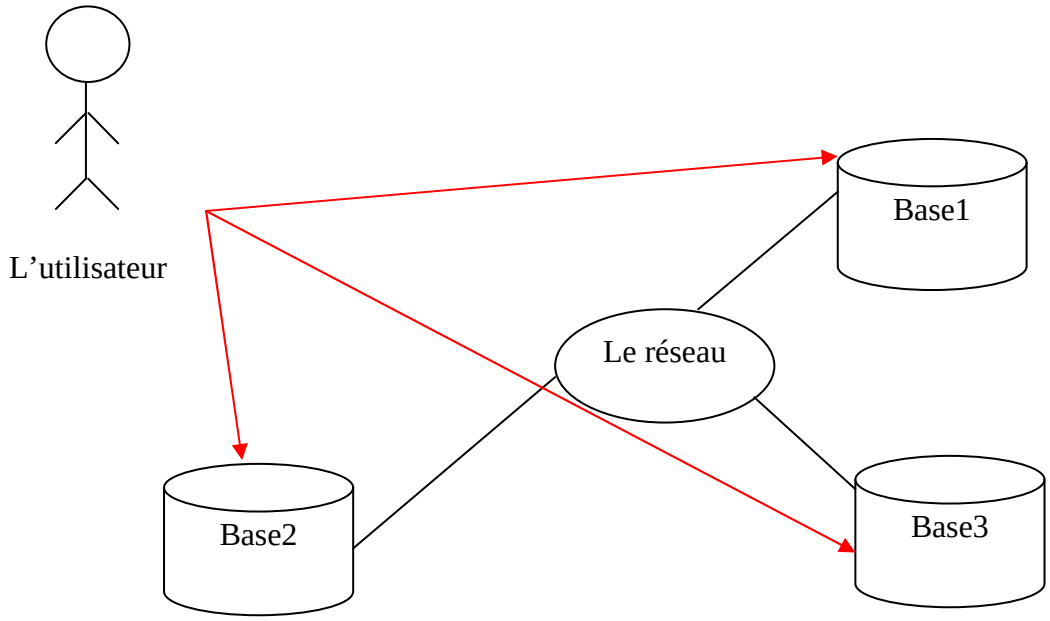

Figure III.3 : La structure globale du système actuel

Le problème traité :

Comment assurer l'intégration des trois sites pour redonner le système développé et ses fonctionnalités plus accessibles par le l'utilisateur ?

La solution proposée :

Puisque les patrons de conception sont généralement destinés à résoudre les problèmes de génération d'objets et d'interaction, plutôt que sur les problèmes à plus grande échelle de l'architecture globale du logiciel, leurs utilisation est une bonne solution pour répondre aux questions posées plus haut.

L'utilisateur accède à l'interface qui sera créée en faisant l'union des trois bases (Base1, Base2, Base3).

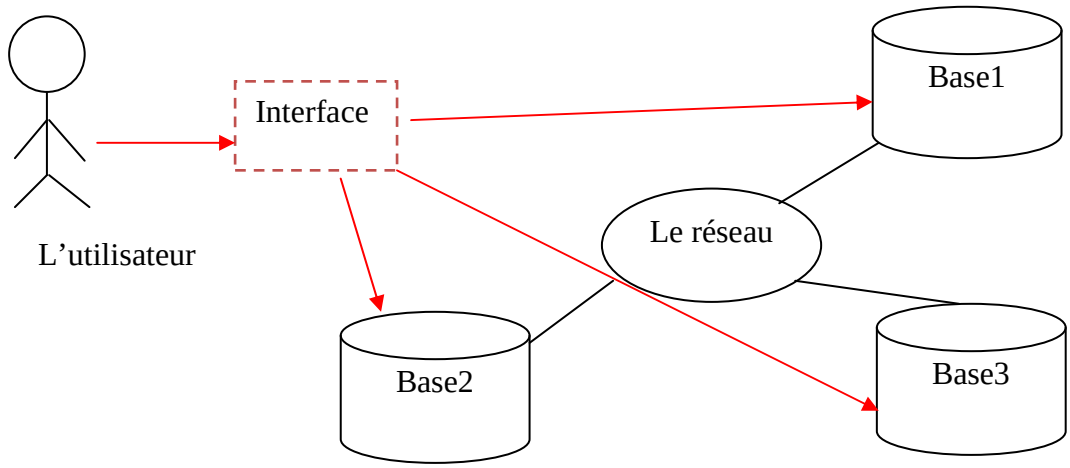

Figure III.4 : La structure globale du système final

III.4. L'analyse :

On présentera dans cette partie les différents patrons de conceptions déjà vu en premier chapitre pour optimiser les lacunes trouvées dans notre cas d'étude.

III.4.1. Patron Adaptateur :

Nom : Adaptateur (Adapter)

Intention :

- Convertir l'interface d'une classe en une autre interface qui est attendue par un client.
- Permet de faire collaborer des classes qui n'auraient pas pu le faire à cause de l'incompatibilité de leurs interfaces

Alias : Empaqueteur (Wrapper), Mariage de convenance

Indications d'utilisation :

- On veut utiliser une classe dont l'interface ne coïncide pas avec celle escomptée
- Prévoir l'ajout de classes non encore connues

Structure :

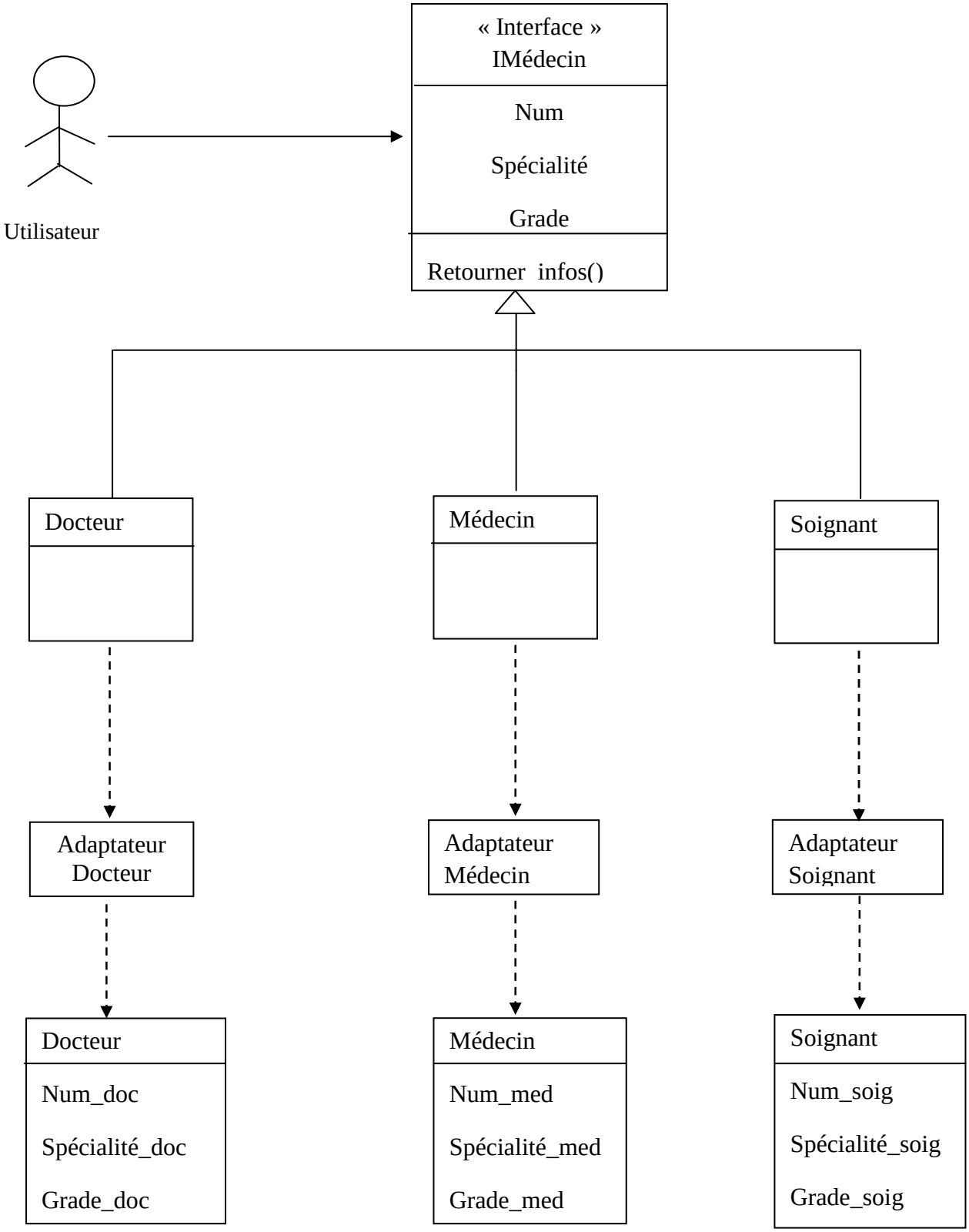

Figure III.5 : Patron Adaptateur

Il permet de convertir l'interface d'une classe en une autre interface que le client attend. L'**Adaptateur** fait fonctionner ensemble des classes qui n'auraient pas pu fonctionner sans lui, à cause d'une incompatibilité d'interfaces.

En effet, dans ce cas le patron adaptateur convertit les interfaces des classes Docteur, Soignant et Médecin en une autre interface adaptateur que l'utilisateur accède directement.

Patterns apparentés

- Adaptateur convertit un protocole dans un autre alors que Décorateur ajoute du protocole
- Bridge est dédié à la séparation entre protocole et implémentation et s'utilise en phase d'analyse, alors que Adaptateur s'applique à des classes existantes pour les connecter à posteriori.
- Commande, de niveau objet, concerne l'exécution et son contrôle : différée, historiée, redirigée, do/undo/redo alors que Adaptateur est de niveau classe et n'est pas concerné pas l'état de l'exécution.

III.4.2. Patron Façade :

Nom : Façade :

Intention

Fournit une interface unifiée pour un sous-système, interface de haut niveau rendant le système plus facile à utiliser (couplage réduit).

Alias : —

Indications d'utilisation

- On souhaite disposer d'une interface simple pour un système complexe.
- Diminuer le couplage entre un sous-système et les clients structuration d'un sous-système en niveaux.

Motivation

- Réduire la complexité d'un système en le découpant en plusieurs sous-systèmes
- Eviter la dépendance entre les clients et les éléments du sous-système

Champs d'application

- Fournir une interface unique pour un système complexe
- Séparer un sous-système de ses clients
- Découper un système en couches (une façade par point d'entrée dans chaque couche)

En génie logiciel, le patron de conception **façade** a pour but de cacher une conception et une interface complexe difficile à comprendre, cette complexité étant apparue « naturellement » avec l'évolution du sous-système en question.

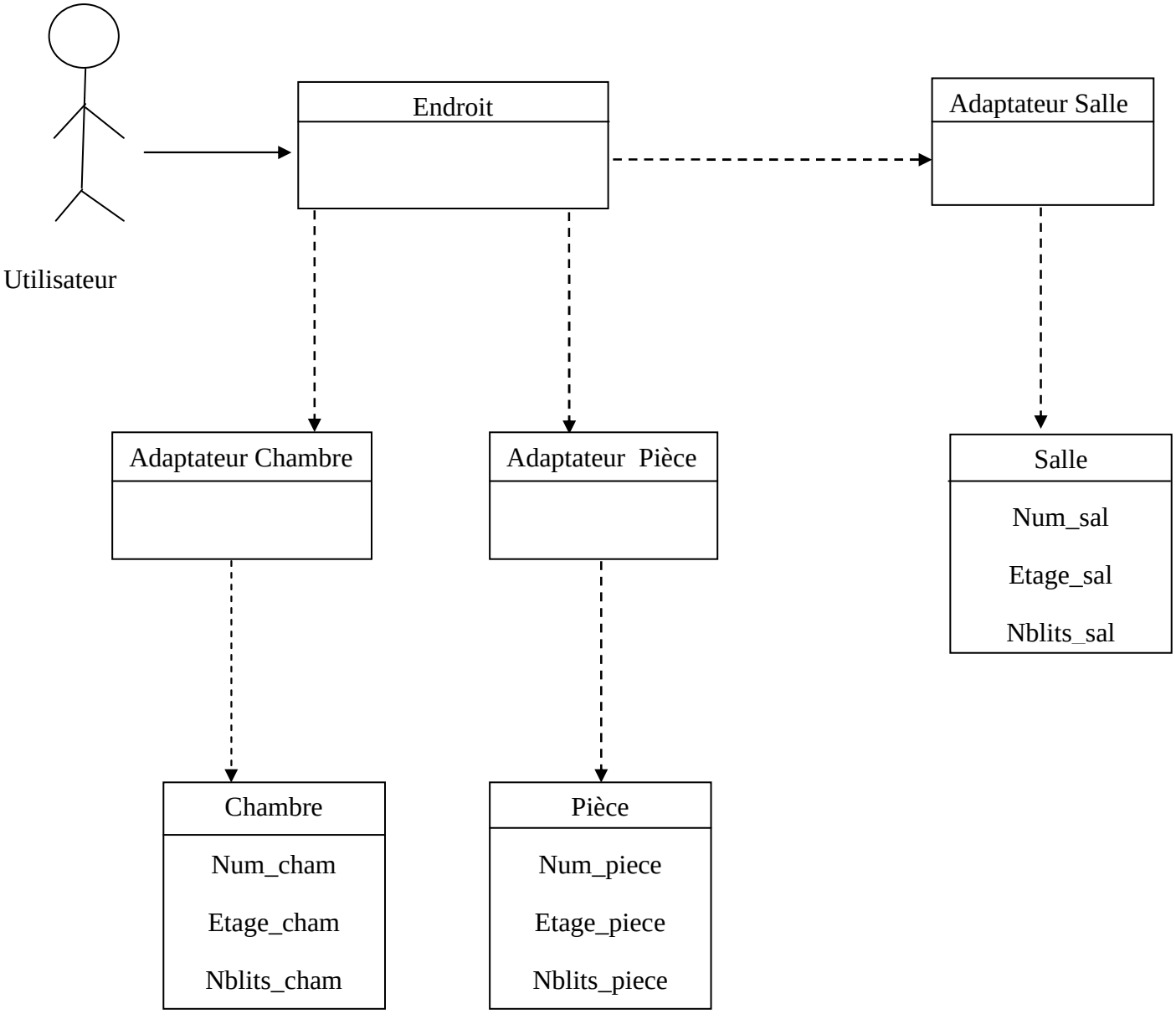

Figure III.6: Utilisation de patron Façade pour connaitre les pièces hospitalières

Dans notre cas, on a un système composé de trois sous systèmes donc un système complexe ce qui permet de simplifier cette complexité en fournissant une interface simple du sous-système.

Participants

- La Façade connaît quelles classes du sous-système sont responsables de telle ou telle requête, et délègue donc les requêtes aux objets appropriés.
- Les classes sous-jacentes à la façade implémentent les fonctionnalités.

- Le nombre de classes n'est pas limité.

Collaborations

- Le client manipule le sous-système en s'adressant à la façade (ou aux éléments du sous-système rendus publics par la façade).
- La façade transmet les requêtes au sous-système après transformation si nécessaire.

Conséquences

- Facilite l'utilisation par la simplification de l'interface
- Diminue le couplage entre le client et le sous-système
- Ne masque pas forcément les éléments du sous-système (un client peut utiliser la façade ou le sous-système)
- Permet de modifier les classes du sous-système sans affecter le client
- Peut masquer les éléments privés du sous-système
- L'interface unifiée présentée par la façade peut être trop restrictive

Implémentation

- Possibilité de réduire encore plus le couplage en créant une façade abstraite et des versions concrètes.
- Les objets de façade sont souvent des singletons.

Patterns associés

Abstract Factory, Mediator, Singleton

III.4.3. Patron Pont:

Considérons une classe représentant la classe de base **lieu de soin**, et ses classes (hôpital, hospice,...). Tous les types de lieu de soin ont des propriétés communes (**Num_lieu**).

Tous les lieux de soin ont des services à accomplir, mais la façon de les faire dépend de l'environnement de chaque lieu de soin.

Ajouter un autre lieu de soin **sanatorium** implique l'ajout de ces services **compartiment1** et **compartiment2** et donc un grand nombre de classes à ajouter ce qui fait un encombrement et un changement au niveau du code de la classe de base **lieu de soin**.

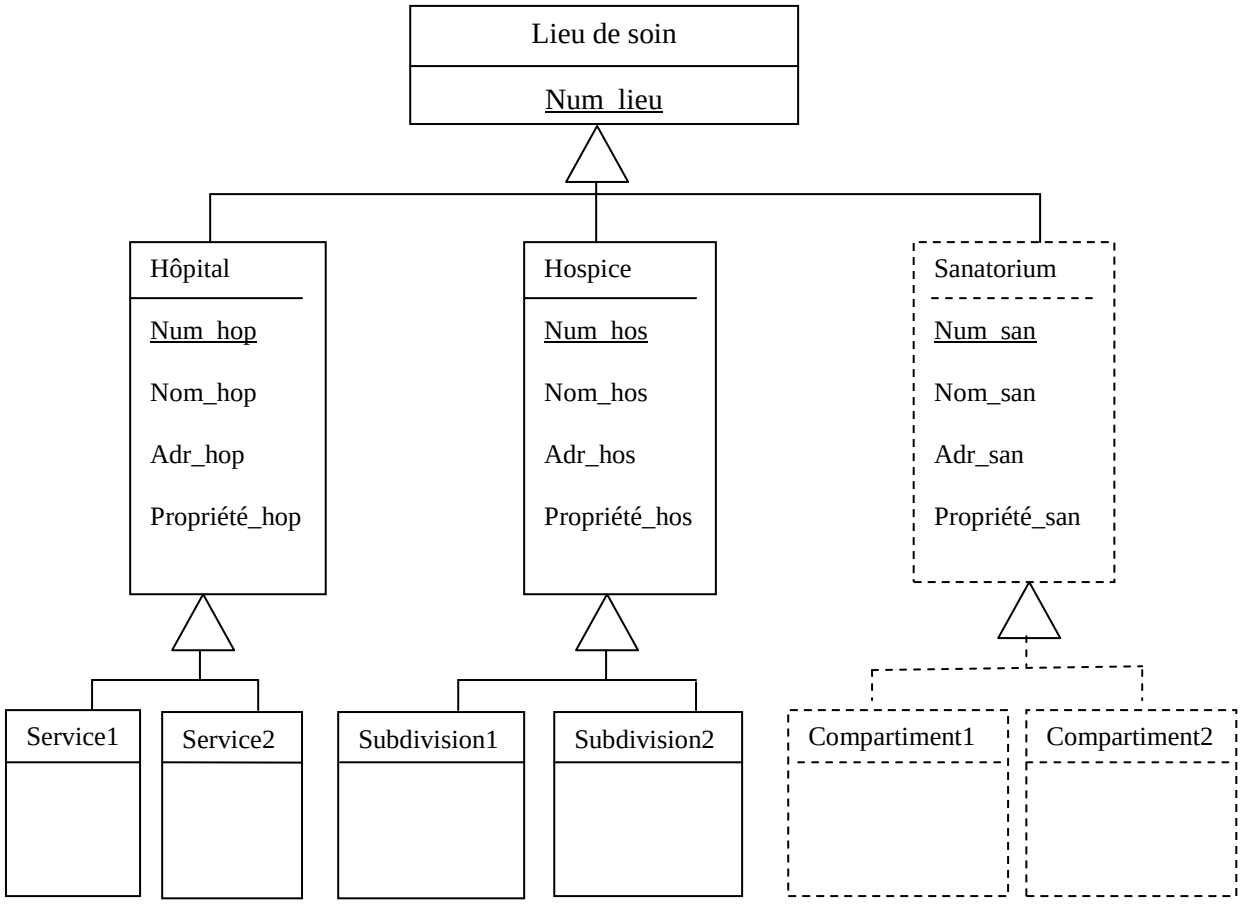

Figure III.7 : Structure globale du système avant application du pont

Intention

Découple une abstraction de sa réalisation afin que les deux puissent être modifiées indépendamment de l'autre

Alias : Poignée/Corps (Handler/Body)

Indications d'utilisation

- Éviter un lien définitif entre une abstraction et son implantation
- Permettre la spécialisation des abstractions et des implantations
- Un changement de l'implantation ne doit pas avoir d'impact sur les clients
- Plusieurs objets partagent la même implantation mais ceci est transparent pour les clients (compteur de références)

Structure :

Le patron de conception Pont suggère de créer une interface séparée pour les primitives de services **endroit**. Cette interface est utilisée par les différents lieux de soin.

Donc l'ajout d'un nouveau lieu de soin est plus facile, il suffit d'insérer **sanatorium** dans **lieu de soin** et insérer les services qu'il offre dans **endroit** sans à porter des modifications au niveau des classes **lieu de soin** et **endroit**.

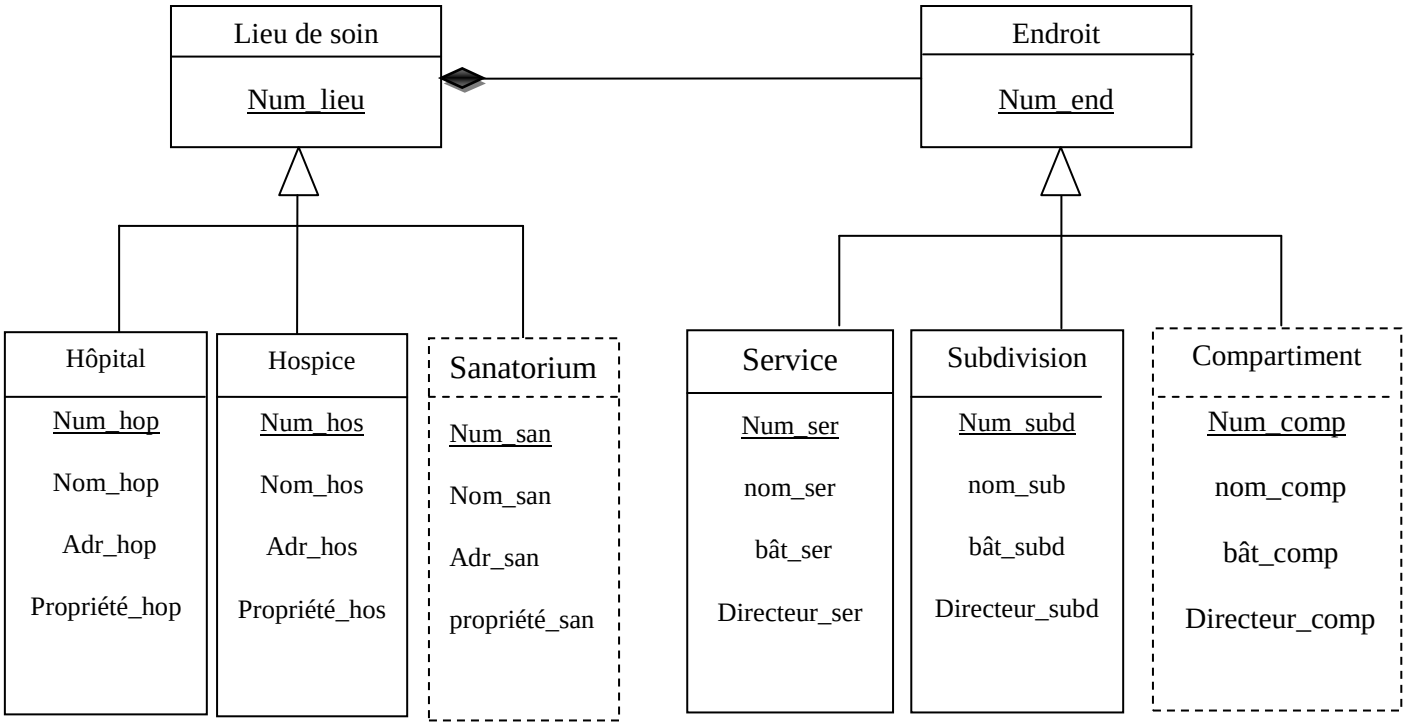

Figure III.8 : Structure globale du système après application du pont

Conséquences :

- Découplage entre abstraction et implantation
- Capacité d'extension accrue
- Dissimulation des détails d'implantation aux clients
- Plusieurs interfaces et implémentations peuvent être couplées/ découplées lors de l'exécution

III.4.4. Patron fabrique abstraite (Abstract Factory) :

Prenons un directeur d'un hôpital dirigeant un certain nombre d'employés, par exemple les infirmiers et les docteurs. Différents employés définissent différentes apparences et comportements pour les composants d'interface utilisateur, Pour être adaptable sur les différents employés, une application ne devrait pas coder en dur ses composants pour un employé particulier. L'instanciation de classes de composants spécifiques à un employé au travers de l'application rendrait difficile la modification de l'employé par la suite.

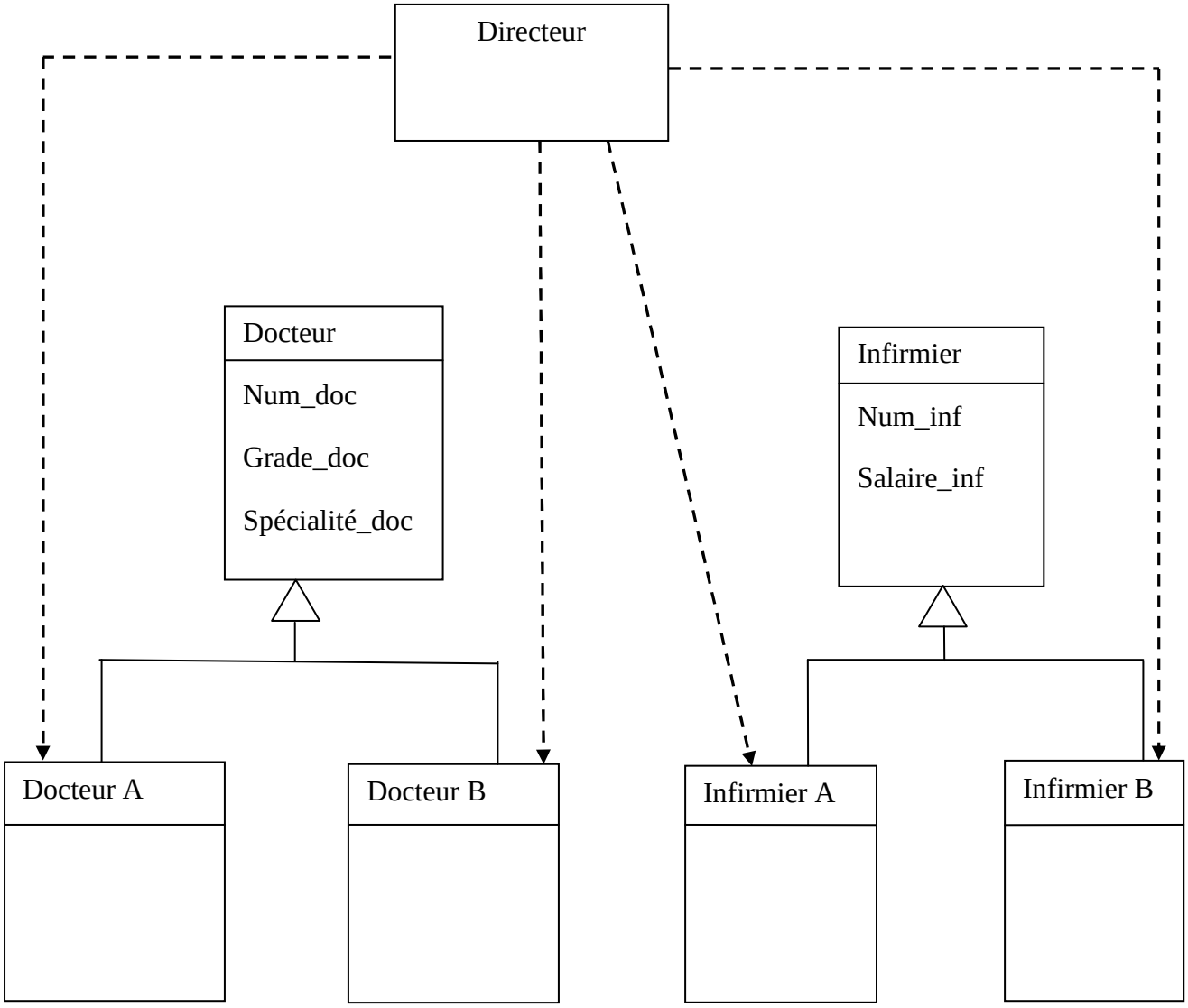

Figure III.9 : Structure du système avant application de la fabrique abstraite

Intention

Fournir une interface pour créer des familles d'objets dépendants ou associés sans connaître leur classe réelle

Alias : Kit, Fabrique abstraite, Usine abstraite

Motivation

Un éditeur qui va produire plusieurs représentations d'un document

Champs d'application

- Indépendance de comment les objets/produits sont créés, composés et représentés
- Configuration d'un système par une instance d'une multitude de familles de produits

- Conception d'une famille d'objets pour être utilisés ensemble et contrôle de cette contrainte
- Bibliothèque fournie avec seulement leurs interfaces, pas leurs implémentations structure

Structure :

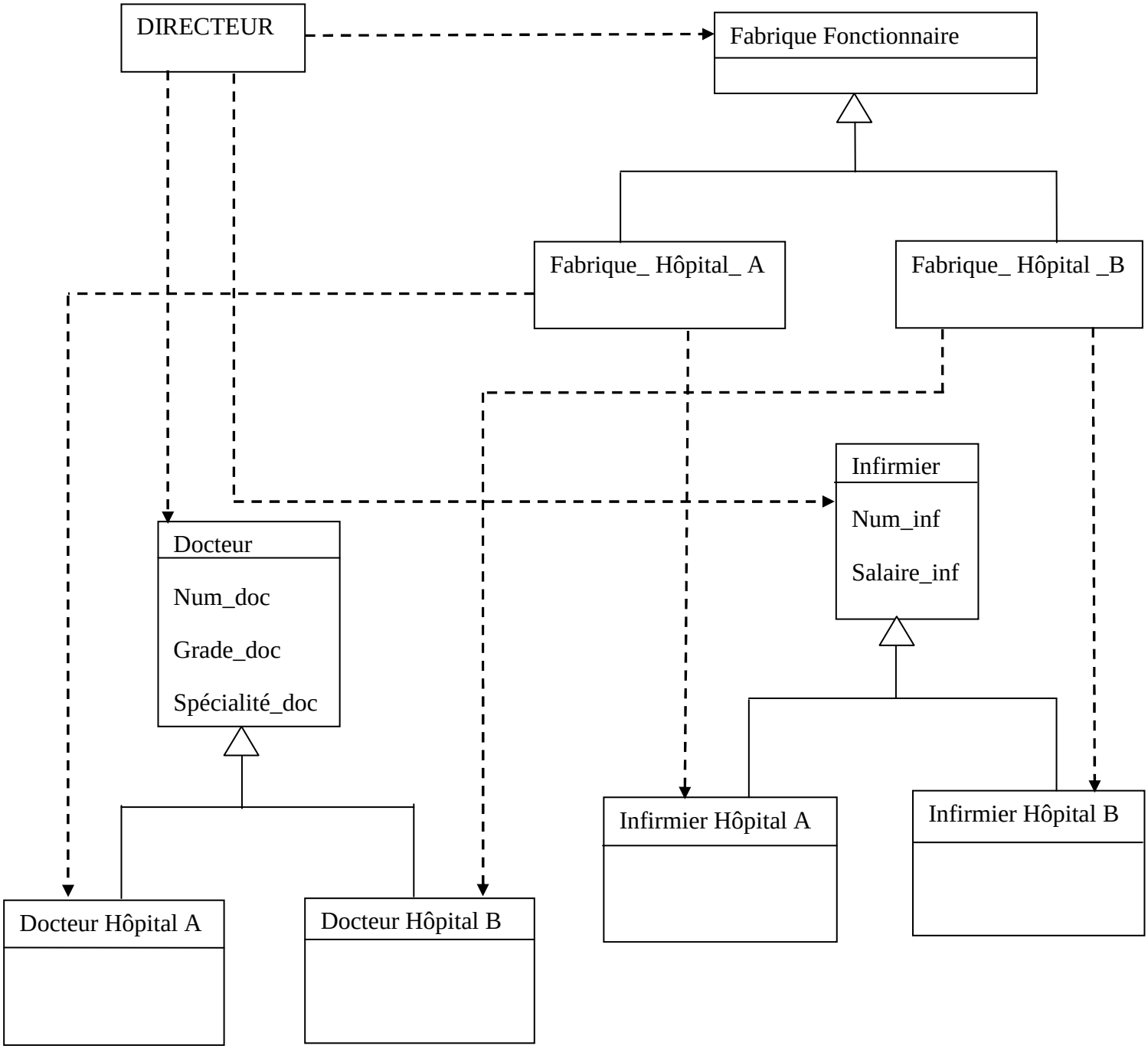

Figure III.10 : Création d'objets à l'aide de la fabrique abstraite

La fabrique fonctionnaire détermine le type de l'objet concret qu'il faut créer (docteur, infirmier), et c'est ici que l'objet est effectivement créé. Cependant, la fabrique retourne un pointeur abstrait sur l'objet concret créé.

Le code du directeur est ainsi isolé de la création de l'objet en l'obligeant à demander à une fabrique de créer l'objet du type abstrait désiré et de lui en retourner le pointeur.

Comme la fabrique retourne uniquement un pointeur abstrait, le code du directeur qui sollicite la fabrique fonctionnaire ne connaît pas et n'a pas besoin de connaître le type concret précis de l'objet qui vient d'être créé. Cela signifie en particulier que :

- Le code du directeur n'a aucune connaissance du type concret, et ne nécessite donc aucune déclaration de classe requis par le type concret. Le code du directeur n'interagit qu'avec la classe abstraite. Les objets concrets sont en effet créés par la fabrique, et le code du directeur ne les manipule qu'avec leur interface abstraite.
- L'ajout de nouveaux types concrets dans le code du directeur se fait en spécifiant l'utilisation d'une fabrique différente, modification qui concerne typiquement une seule ligne de code (une nouvelle fabrique crée des objets de types concrets différents, mais renvoie un pointeur du même type abstrait, évitant ainsi de modifier le code du directeur). C'est beaucoup plus simple que de modifier chaque création de l'objet dans le code du directeur.

Participants :

- Fabrique fonctionnaire déclare l'interface des opérations qui créent des objets abstraits (infirmier, docteur).
- Fabrique_hôpital_ A et fabrique_hôpital_B implémentent les opérations qui créent les objets concrets.
- Docteur et infirmier déclarent des interfaces pour un type d'objets
- Le directeur utilise seulement les interfaces déclarées par la fabrique fonctionnaire et par les classes docteur et infirmier.

Collaborations

Normalement, une seule instance de fabrique **concrète** est créée à l'exécution. Cette fabrique crée les objets avec une implémentation spécifique. Pour créer différents sortes d'objets, les clients doivent utiliser différentes fabriques concrètes.

La fabrique abstraite défère la création des objets à ses sous-classes concrètes.

Conséquences

- Isolation des classes concrètes
- Échange facile des familles d'employés (docteur, infirmier)

- Séparation des classes concrètes (fabrique_hôpital_A, fabrique_hôpital_B), des classes clients (directeur).
- Les noms des classes employés n'apparaissent pas dans le code directeur
- Le processus de création est clairement isolé dans une classe
- La mise en place de nouveaux employés dans la fabrique fonctionnaire n'est pas aisée

Implémentation

- Les fabriques sont souvent des singletons
- Ce sont les sous-classes concrètes qui font la création, en utilisant le plus souvent une Factory Method
- Si plusieurs familles sont possibles, la fabrique concrète utilise Prototype

Patterns associés

- Souvent implémentés en utilisant des méthodes de fabrication (pattern Fabrique)
- Façade : interface unifiée de manipulation d'un système complexe
- Une fabrique abstraite est souvent un Singleton
- Pattern Pont (Bridge) séparation couche d'objets d'abstraction / couche d'objets d'implantation

Conclusion :

On a dévoilé dans ce chapitre les méthodes de résolution de problèmes récurrents (sans respect au modèle MVC, avec respect au modèle MVC) ensuite on a présenté notre cas d'étude et on a extrait quelques patrons de conception comme solutions aux difficultés rencontrées.

Le dernier chapitre sera consacré à la réalisation.

Chapitre IV

Réalisation

Introduction :

Après avoir exposé dans le chapitre précédent les patrons de conception utilisés selon notre cas d'étude, nous allons présenter dans ce dernier chapitre l'environnement de développement ainsi que les outils qu'on a utilisé pour atteindre notre réalisation.

IV.1. Outils de développement :

IV.1.1. Oracle 11g :

IV.1.1.1. Définition : [18] [19]

Oracle est un SGBD (système de gestion de bases de données) édité par la société du même nom (Oracle Corporation), leader mondial des bases de données.

La société *Oracle Corporation* a été créée en 1977 par Lawrence Ellison, Bob Miner, et Ed Oates. Elle s'appelle alors *Relational Software Incorporated (RSI)* et commercialise un Système de Gestion de Bases de données relationnelles (SGBDR ou RDBMS pour *Relational Database Management System*) nommé *Oracle*.

Oracle est écrit en langage C et est disponible sur de nombreuses plates-formes matérielles (plus d'une centaine) dont AIX (IBM), Solaris (Sun), HP/UX (Hewlett Packard), Windows NT (Microsoft).

Oracle 11g est la première base de données conçue pour le « Grid Computing ». Que ce soit sur des serveurs Windows, Linux ou UNIX, Oracle Database 11*g* pulvérise tous les records de performances et d'évolutivité. Autre caractéristique clé, la base de données Oracle permet de migrer d'un serveur unique vers le Grid Computing sans avoir à modifier une seule ligne de code, d'où un ROI quasi immédiat.

Oracle Database 11*g* permet de meilleurs résultats grâce notamment à l'automatisation des tâches administratives et à des fonctionnalités de sécurité et de conformité réglementaire sans équivalent sur le marché. Avec Real Application Clusters (RAC), Oracle Database 11*g* assure la fonction de haute disponibilité. Enfin, avec une large gamme de versions et des coûts d'exploitation réduits, la solution phare d'Oracle s'impose comme le choix privilégié de toutes les entreprises et ce quel que soit leur taille.

Figure IV.1 : Interface d'Oracle 11g

IV.1.1.2. Les fonctionnalités d'Oracle : [19]

Oracle est un SGBD permettant d'assurer :

- La définition et la manipulation des données
- La cohérence des données
- La confidentialité des données
- L'intégrité des données
- La sauvegarde et la restauration des données
- La gestion des accès concurrents

IV.1.1.3. Les composants d'Oracle : [19]

Outre la base de données, la solution Oracle est un véritable environnement de travail constitué de nombreux logiciels permettant notamment une administration graphique d'Oracle, de s'interfacer avec des produits divers et d'assistants de création de bases de données et de configuration de celles-ci.

On peut classer les outils d'Oracle selon diverses catégories :

- Les outils d'administration
- Les outils de développement
- Les outils de communication
- Les outils de génie logiciel
- Les outils d'aide à la décision

IV.1.1.4. Outils de développement d'Oracle : [19]

Oracle propose également de nombreux outils de développement permettant d'automatiser la création d'applications s'interfaçant avec la base de données. Ces outils de développement sont :

- Oracle Designer
- Oracle Developer
- SQL*Plus : Une interface interactive permettant d'envoyer des requêtes SQL et PL/SQL à la base de données. SQL*Plus permet notamment de paramétrer l'environnement de travail (formatage des résultats, longueur d'une ligne, nombre de lignes par page, ...)
- Oracle Developper : Il s'agit d'une suite de produits destinés à la conception et à la création d'applications client-serveur. Il est composé de 4 applications :
  - Oracle Forms (anciennement SQL*Forms) : Un outil permettant d'interroger la base de données de façon graphique sans connaissances préalables du langage SQL. SQL*Forms permet ainsi de développer des applications graphiques (fenêtres, formulaires, ...) permettant de sélectionner, modifier et supprimer des données dans la base.
  - Oracle Reports (SQL*ReportWriter) : Un outil permettant de réaliser des états.
  - Oracle Graphics : Un outil de génération automatique de graphiques dynamiques pour présenter graphiquement des statistiques réalisées à partir des données de la base.
  - Procedure Builder : Un outil permettant de développer des procédures, des fonctions et des packages.

IV.1.1.5. Architecture du SGBD Oracle : [19]

Une base de données Oracle est constituée de plusieurs éléments :

- Des processus chargés en mémoire sur le serveur
- Des fichiers physiques stockés sur le serveur
- D'un espace mémoire sur le serveur appelé *SGA* (*System Global Area*)

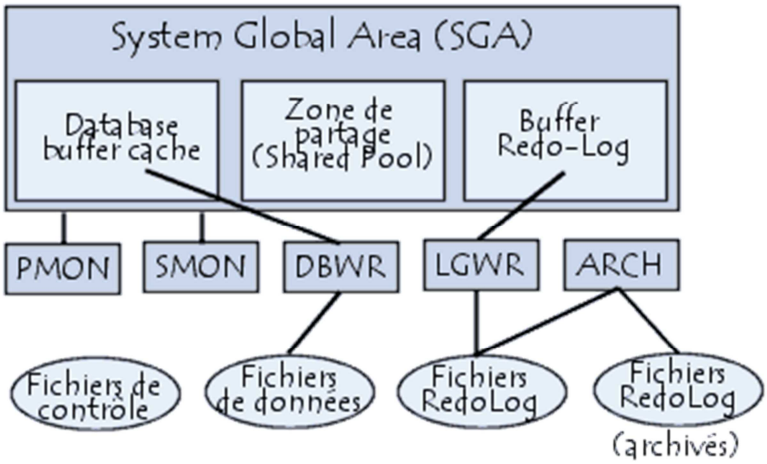

Figure IV.2 : architecture d'Oracle

IV.1.2. PL/SQL Developer : [19]

Le langage PL/SQL est un langage L4G (entendez par ce terme un langage de quatrième génération), fournissant une interface procédurale au SGBD Oracle. Le langage PL/SQL intègre parfaitement le langage SQL en lui apportant une dimension procédurale.

En effet, le langage SQL est un langage déclaratif non procédural permettant d'exprimer des requêtes dans un langage relativement simple. En contrepartie il n'intègre aucune structure de contrôle permettant par exemple d'exécuter une boucle itérative.

Ainsi le langage PL/SQL permet de manipuler de façon complexe les données contenues dans une base Oracle en transmettant un bloc de programmation au SGBD au lieu d'envoyer une requête SQL. De cette façon les traitements sont directement réalisés par le système de gestion de bases de données. Cela a pour effet notamment de réduire le nombre d'échanges à travers le réseau et donc d'optimiser les performances des applications.

D'autre part le langage PL/SQL permet de faire appel à des procédures externes, c'est-à-dire des procédures écrites dans un autre langage (de troisième génération, généralement le langage C).

Le langage PL/SQL permet de définir un ensemble de commandes contenues dans ce que l'on appelle un "bloc" PL/SQL. Un bloc PL/SQL peut lui-même contenir des sous-blocs. La syntaxe PL/SQL est simple et lisible.

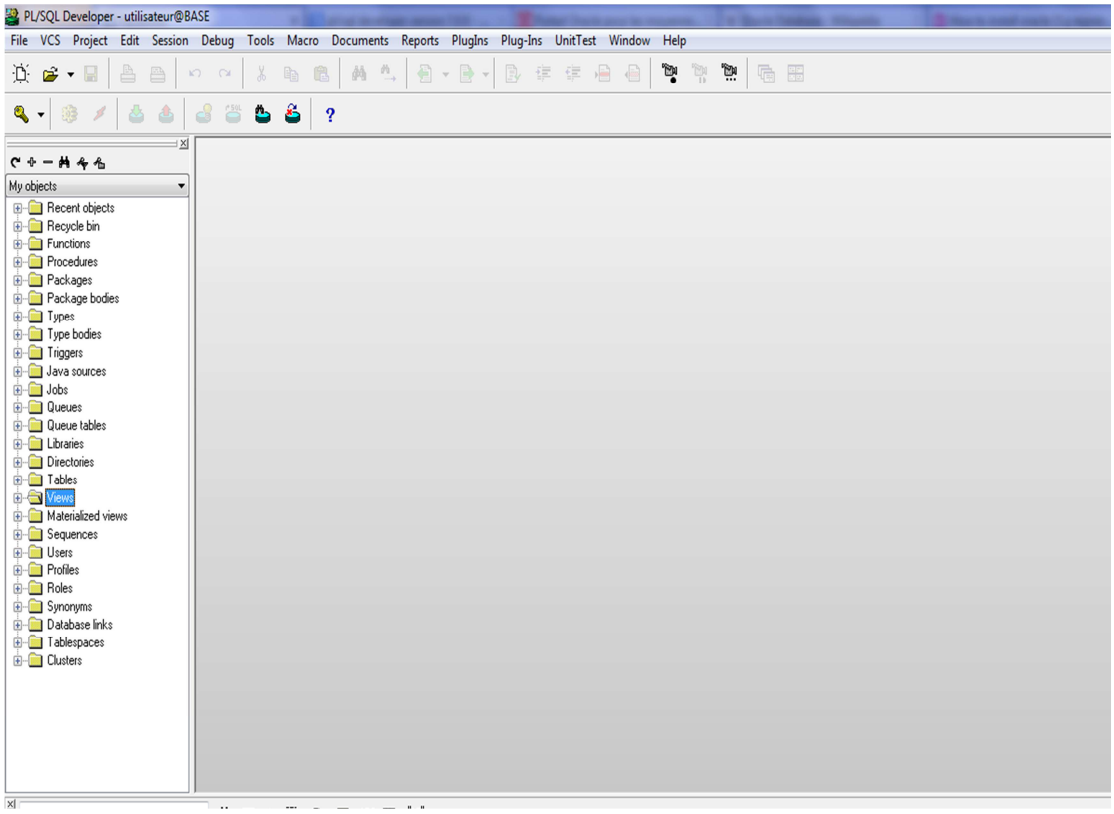

Figure IV.3 : Interface de PL/SQL Developer

IV.1.3. NetBeans IDE 7.2.1 :

NetBeans est un environnement de développement intégré (IDE) pour développer principalement avec Java , mais aussi avec d'autres langues, en particulier PHP , C / C + + et HTML5 . C'est aussi une plate-forme que demande Framework pour les applications Java Desktop et autres. L'EDI NetBeans est écrit en Java et peut fonctionner sur Windows, Linux, Solaris et autres plateformes supportant un compatible JVM (Java Virtuel Machine).

La plate-forme NetBeans permet aux applications d'être développées à partir d'un ensemble de modules composants logiciels appelés *modules.* Les applications basées sur la plate-forme NetBeans (y compris l'EDI NetBeans lui-même) peuvent être étendues par des développeurs tiers .

L'équipe NetBeans soutient activement le produit et consulte longues suggestions de la communauté au sens large. Chaque version est précédée par une période de test de la Communauté et des commentaires.

La plate-forme offre des services réutilisables communs aux applications de bureau, permettant aux développeurs de se concentrer sur la logique spécifique à leur application. Parmi les fonctionnalités :

- Gestion de l'interface utilisateur (par exemple, les menus et barres d'outils)
- Gestion des paramètres de l'utilisateur
- La gestion du stockage (sauvegarde et le chargement tout type de données)
- La gestion de la fenêtre
- Cadre Assistant (supports étape par étape dialogues)
- NetBeans Bibliothèque visuelle
- Outils de développement intégrés

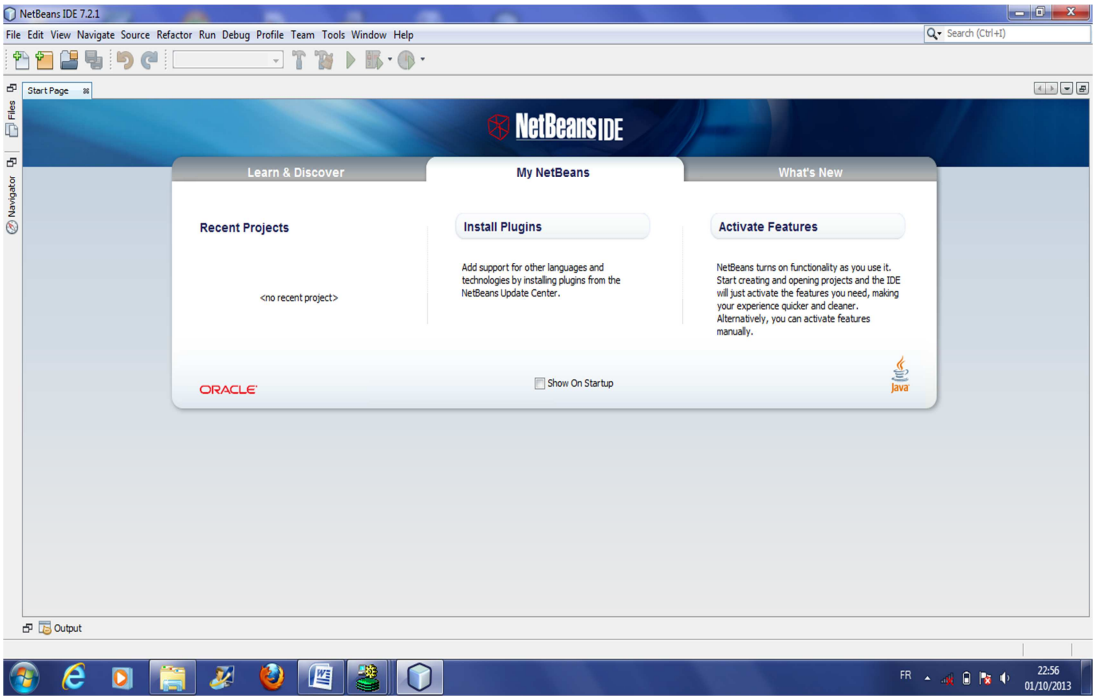

Figure IV.4 : l'interface de NetBeans 7.2.1

IV.2. Les bases de données réparties :

IV.2.1. Définition d'une base de données : [3]

Une base de données est un conteneur servant à stocker des données: des renseignements bruts tels que des chiffres, des dates ou des mots, qui peuvent être retraités par des moyens informatiques en vue de produire une information; elle est la pièce centrale d'un dispositif informatique dit système de base de données qui dirige la collecte le stockage, le retraitement et l'utilisation de données.

IV.2.2. Définition d'une base de données répartie : [3]

Une base de données répartie (BDR) est une base de données dont différentes parties sont stockées sur des sites, généralement géographiquement distants, reliés par un réseau. La réunion de ces parties forme la base de données répartie.

Un système de bases de données réparties ne doit donc en aucun cas être confondu avec un système dans lequel les bases de données sont accessibles à distance (selon le principe client serveur). Il ne doit non plus être confondu avec un système multi base. Dans ce dernier cas, chaque utilisateur accède à différentes bases de données en spécifiant leur nom et adresse, et le système se comporte alors simplement comme un serveur de BD et n'apporte aucune fonctionnalité particulière à la répartition.

Au contraire, un système de bases de données réparties est suffisamment complet pour décharger les utilisateurs de tous les problèmes de concurrence, fiabilité, optimisation de requêtes ou transaction sur des données gérées par différents SGBD sur plusieurs sites.

IV.2.3. Raisons de répartition des données : [20]

- Les pressions pour la distribution :
  - Il devient impératif de décentraliser l'information,
  - Augmentation du volume de l'information,
  - Augmentation du volume des transactions.
  - Besoin de serveurs de BDs qui fournissent un bon temps de réponse sur des gros volumes de données.
- Prédiction d'accroissement:
  - Vitesse des microprocesseurs,
  - Capacité des DRAM,
  - Débit des disques.
- Pour améliorer le débit des E/Ss :
  - Partitionnement des données,
  - Accès parallèle aux données,
  - Utiliser plusieurs nœuds (avec un bon coût/ performance), et les faire communiquer par un réseau.

IV.2.4. Buts de répartition des bases de données : [20]

Les bases de données réparties ont une architecture plus adaptée à l'organisation des entreprises décentralisées.

- Plus de fiabilité : les bases de données réparties ont souvent des données répliquées. La panne d'un site n'est pas très importante pour l'utilisateur, qui s'adressera à autre site.
- Meilleures performances : réduire le trafic sur le réseau est une possibilité d'accroître les performances. Le but de la répartition des données est de les rapprocher de l'endroit où elles sont accédées. Répartir une base de données sur

plusieurs sites permet de répartir la charge sur les processeurs et sur les entrées/ sorties.

- Faciliter l'accroissement: l'accroissement se fait par l'ajout de machines sur le réseau.

IV.2.5. SGBD réparti : [3] [20]

Le système de gestion de base de données (SGBD) est une suite de programmes qui manipule la structure de la base de données et dirige l'accès aux données qui y sont stockées. Il sert d'intermédiaire entre la base de données et ses usagers.

Une base de données centralisée est gérée par un seul SGBD, est stockée dans sa totalité à un emplacement physique unique et ses divers traitements sont confiés à une seule et même unité de traitement. Par opposition, une base de données distribuée est gérée par plusieurs processeurs, sites ou SGBD.

IV.3. Les liens de bases de données :

IV.3. 1. Définition : [19]

Un lien de base de données est un pointeur qui définit un chemin de communication unidirectionnelle à partir d'un serveur de base de données Oracle à un autre serveur de base de données. Le pointeur de lien est en fait défini comme une entrée dans une table de dictionnaire de données. Pour accéder au lien, vous devez être connecté à la base de données locale qui contient l'entrée du dictionnaire de données.

Une connexion de liaison de base de données est à sens unique dans le sens où un client connecté à la base locale A peut employer un lien stocké dans la base A pour accéder aux informations en base de données distante B, mais les utilisateurs connectés aux bases de données B ne peuvent pas utiliser le même lien pour accéder aux données base de données A. Si les utilisateurs locaux de la base B veulent accéder aux données de la base A, alors ils doivent définir un lien qui est stocké dans le dictionnaire de données de base B.

IV. 3.2. Avantages d'utilisation des liens de bases de données : [19]

Le grand avantage de liens de base de données est qu'ils permettent aux utilisateurs d'accéder aux objets d'un autre utilisateur dans une base de données distante pour qu'elles soient limitées par le jeu de privilèges du propriétaire de l'objet. En d'autres termes, un utilisateur local peut accéder à un lien vers une base de données à distance sans avoir à être un utilisateur sur la base de données distante.

IV.3.3. La création de liens : [19]

Pour créer un DBLink, il faut que le tnsname.ora soit correctement renseigné concernant le SID de la base distante, et cela à l'aide de l'instruction suivante :

CREATE DATABASE LINK 'NOM-LIEN' connect to 'user' identified by 'password' using 'nom_BDD'

Pour pouvoir se connecter à la base1, à partir de la base « base » on créera un lien de BDD « lien1 » à l'aide du code suivant :

CREATE DATABASE LINK lien1 connect to user1 identified by user1 using base1

Parfois, pour une opération ponctuelle ou en raison du fait que l'on n'ait pas accès au système d'exploitation pour éditer le tnsname.ora, il est possible de créer un DBLink directement en utilisant la chaine de connexion complète :

Exemple:

```
Create database link base connect to user2 identified by user2
'(DESCRIPTION =
(ADRESS_LIST =
(ADRESS = (PROTOCOL = TCP) (HOST = 192.168.1.29) (PORT = 1521))
)
(CONNECT_DATA =
(SID = base2)
(SERVEUR = DEDICATED)
)
)' ;
```

La figure suivante montre les DBLinks créés selon notre cas d'étude :

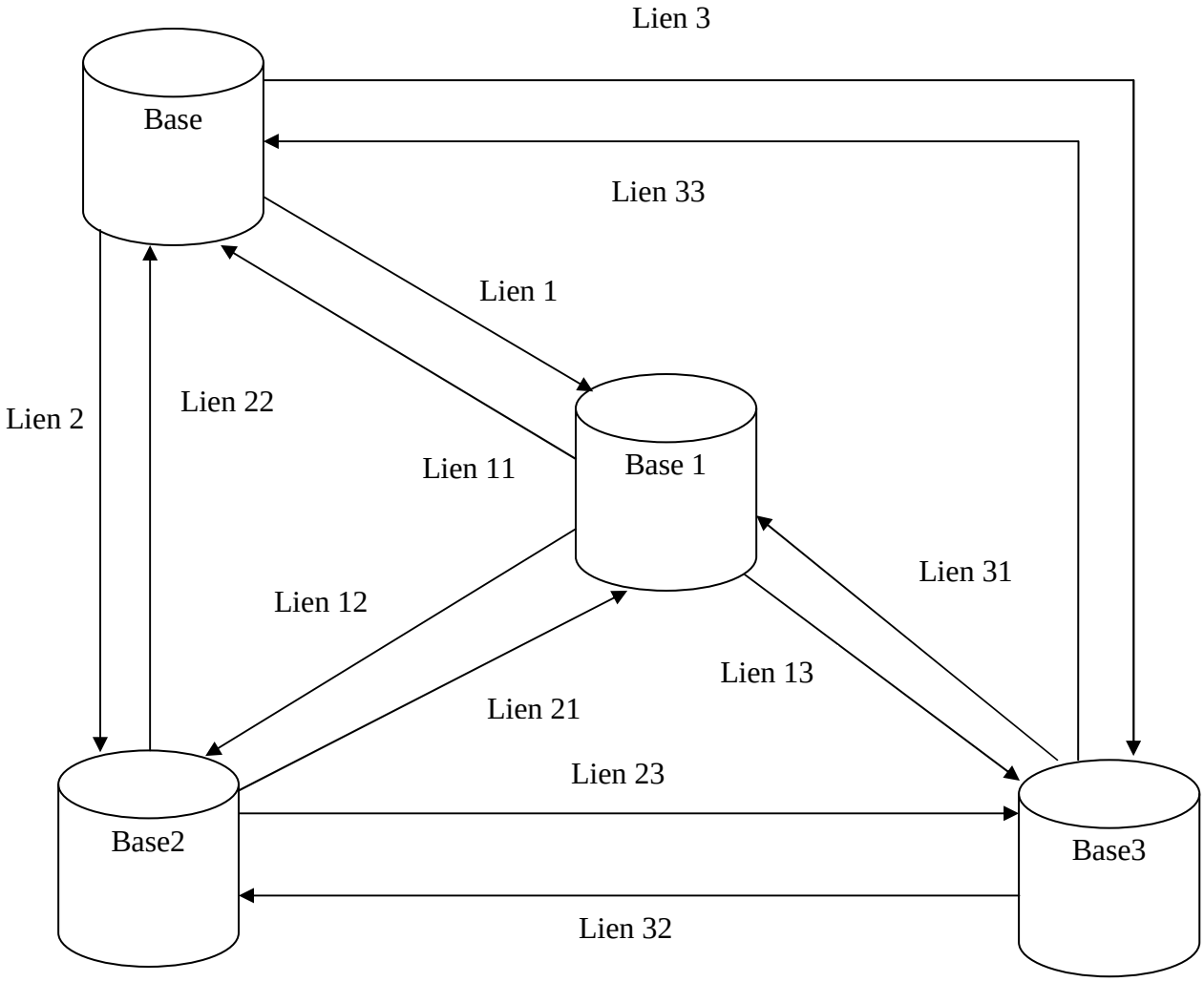

Figure IV.5 : schéma des liens de base de données

IV.3.4. Les types de liens: [19]

Oracle Database vous permet de créer privés, publics et mondial liens de bases de données. Ces types de liens de base diffèrent selon laquelle les utilisateurs sont autorisés à accéder à la base de données distante:

- **Privé :** Crée un lien dans un schéma spécifique de la base de données locale. Seul le propriétaire d'un lien de base de données privé ou sous-programmes PL / SQL dans le schéma peut utiliser ce lien pour accéder à des objets de base de données dans la base de données distante correspondante.

L'utilisateur peut voir les DBLinks existants à travers les requêtes suivantes :

- `SELECT * FROM DBA_DB_LINKS;`
- `SELECT * FROM ALL_DB_LINKS;`
- `SELECT * FROM USER_DB_LINKS;`

- **Publique:** Crée un lien base de données à l'échelle. Tous les utilisateurs et sous-programmes PL / SQL dans la base de données peuvent utiliser ce lien pour accéder à des objets de base de données dans la base de données distante correspondante.
  Utilisateur appelé PUBLIC peut voir les données de propriété à travers des vues présentées par des liens de bases de données privées.
- **Mondial :** Crée un lien échelle du réseau. Quand un réseau Oracle utilise un serveur d'annuaire, le serveur d'annuaire crée et gère automatiquement les liens de bases de données globales (comme les noms de service nets) pour chaque base de données Oracle dans le réseau. Les utilisateurs et les sous-programmes PL / SQL dans une base de données peuvent utiliser un lien mondial pour accéder aux objets de la base de données distante correspondante.

  L'utilisateur peut voir les données de la même façon que dans PUBLIC

IV.3.5. Utilisation des liens : [19]

Une fois que vous avez créé un lien de base de données, il vous permet de faire une requête sur un, voire plusieurs, objet(s) distant(s) au sein d'une session. Il suffit d'avoir les droits d'accès pour se connecter à la base distante (connaitre le nom de la base distante, l'utilisateur distant et son mot de passe) pour pouvoir exécuter des instructions SQL. Par exemple, pour accéder à l'objet distant employé (créé à la base1) à partir de la base2 on utilise le lien lien21 (déjà créé en utilisant la base1 comme base distante, user1/user1 comme utilisateur/mot de passe) vous pouvez émettre:

SELECT * FROM employé @ lien21;

Construire des noms d'objets correctement formés à l'aide de liens de base de données est un aspect essentiel de la manipulation de données dans les systèmes distribués.

On montrera dans ce qui suit comment on a pu utiliser les liens de bases de données selon notre cas d'étude à l'aide des patrons de conception suivants :

IV.3.5.1. Patron Adaptateur :

L'objet « adapt » illustre les résultats de l'utilisation de patron adaptateur. Pour que l'utilisateur accède aux objets de la BDD « Base » en utilisant les liens de BDD (Lien1, Lien2, Lien3) on émettra la requête suivante sur NetBeans IDE 7.2.1:

Select * from utilisateur.adapt ;

Et les résultats de la requête sont illustrés dans la figure suivante :

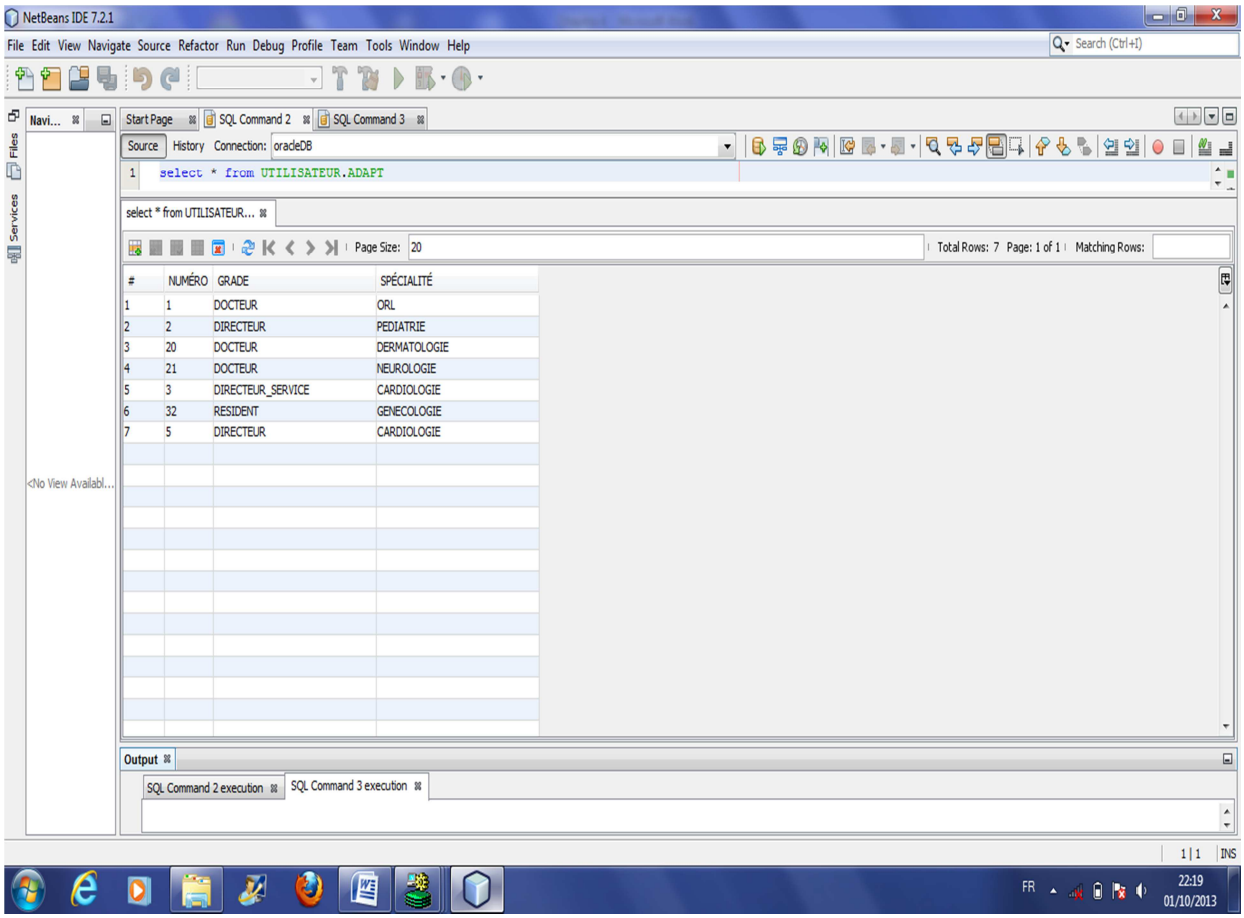

Figure IV.6 : Résultats d'application de patron Adaptateur

IV.3.5.2. Patron Façade :

L'objet « facade » illustre les résultats de l'utilisation de patron facade. Pour que l'utilisateur accède aux objets de la BDD « Base » en utilisant les liens de BDD (Lien1, Lien2, Lien3) on émettra la requête suivante sur NetBeans IDE 7.2.1:

Select * from utilisateur. Façade ;

Et les résultats de la requête sont illustrés dans la figure suivante :

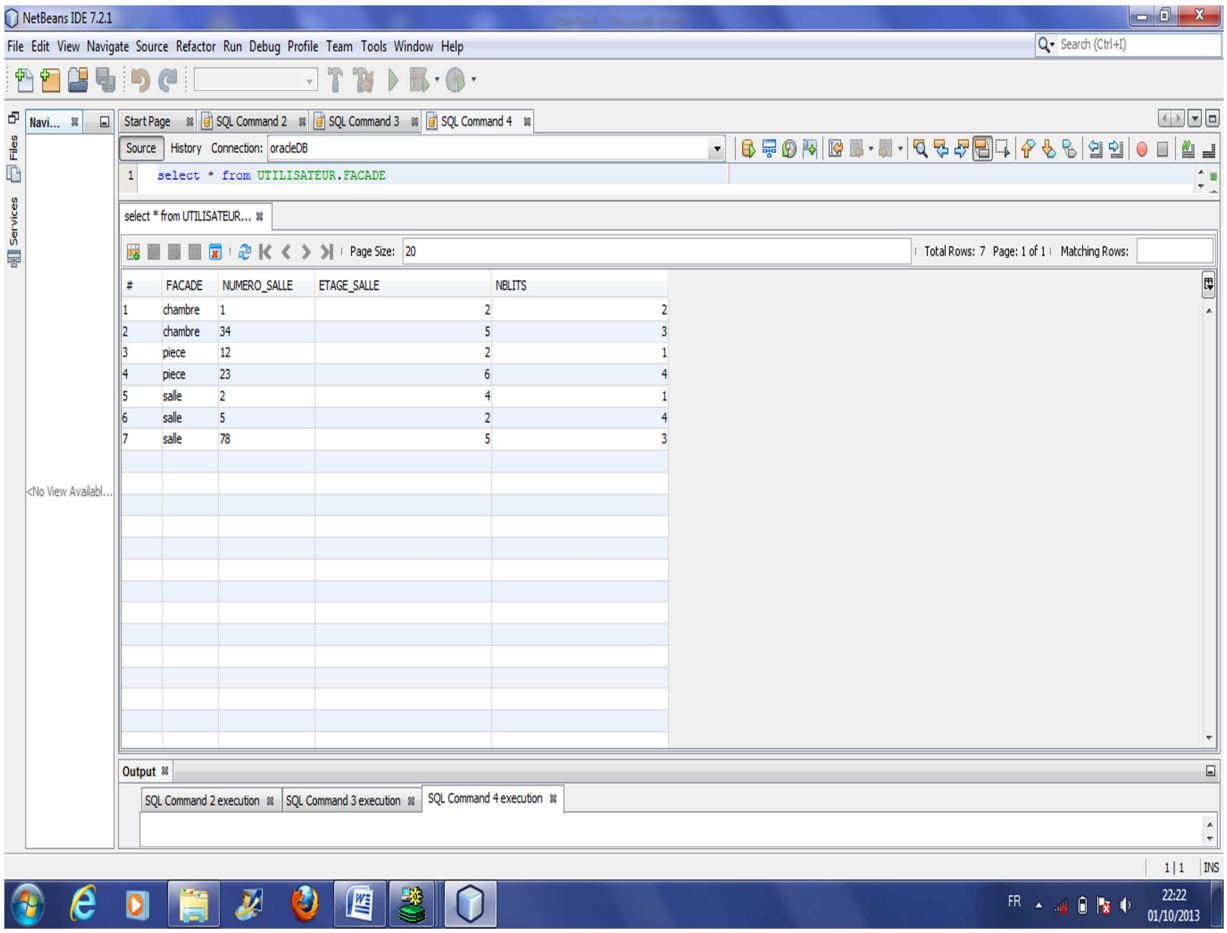

Figure IV.7 : Résultats de l'application de patron façade

IV.3.5.3. Patron pont :

Les objets « lieu_soin » et « endroit » illustrent les résultats de l'utilisation de patron pont. Pour que l'utilisateur accède aux objets de la BDD « Base » en utilisant les liens de BDD (Lien1, Lien2, Lien3).

On émettra la requête suivante sur NetBeans IDE 7.2.1 pour afficher les lieux de soin (hôpitaux) :

Select * from utilisateur.lieu_soin ;

Et les résultats de la requête sont illustrés dans la figure suivante :

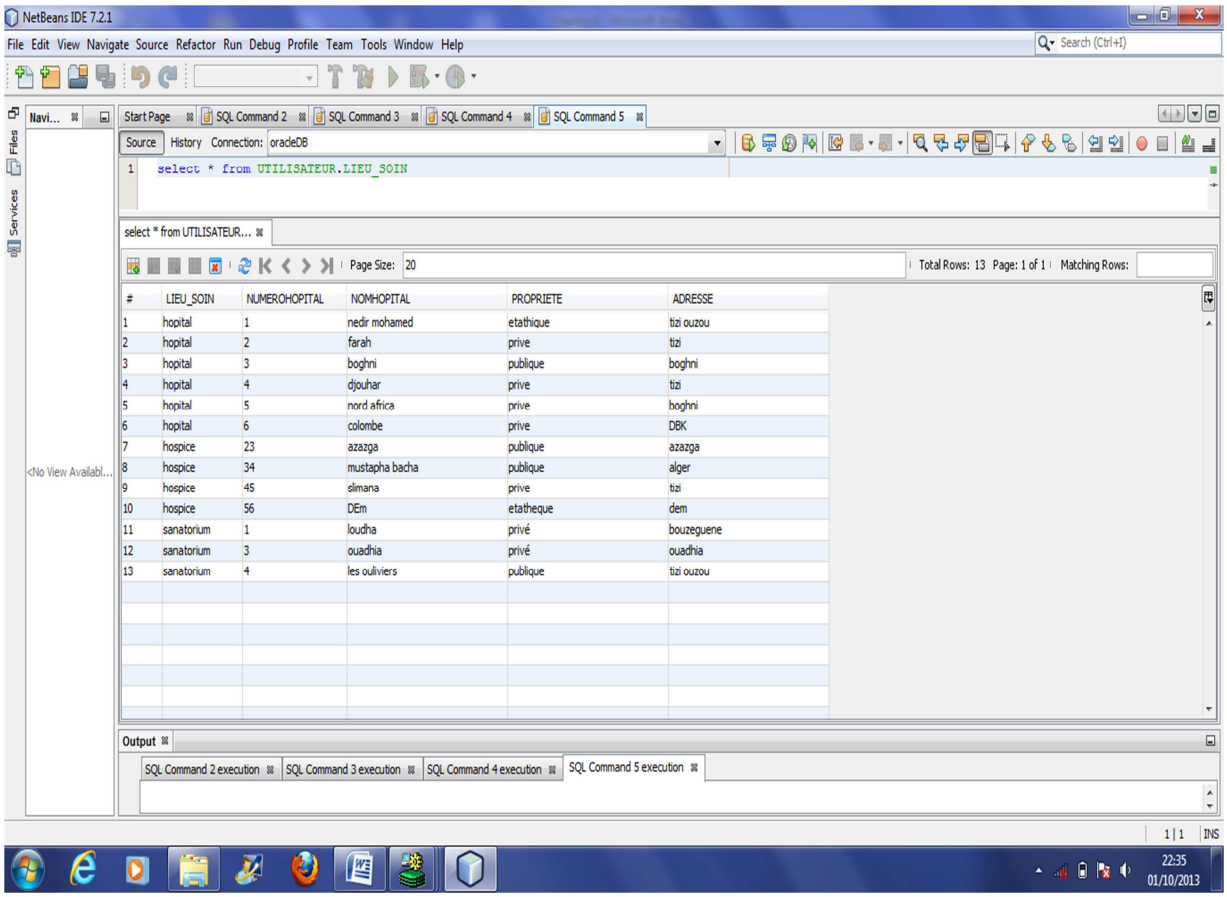

Figure IV.8 : Résultats d'application de patron pont pour les lieux de soin

On émettra la requête suivante sur NetBeans IDE 7.2.1 pour afficher les endroits (services) des lieux de soin :

Select * from utilisateur. Endroit ;

Et les résultats de la requête sont illustrés dans la figure suivante :

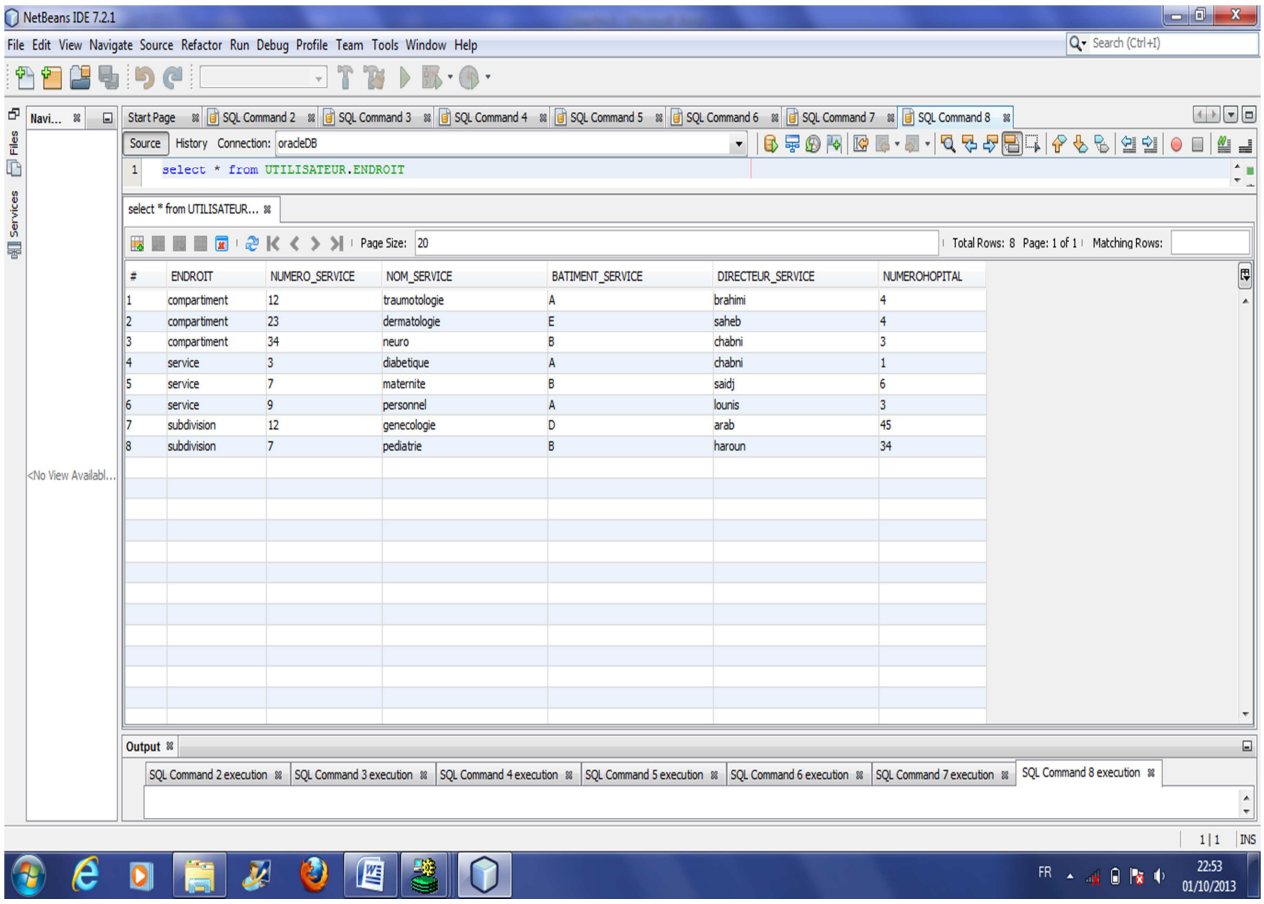

Figure IV.9 : Résultats d'application de patron pont pour les endroits

IV.3.5.3. Patron fabrique abstraite :

L'objet « Fabrique » illustre les résultats de l'utilisation de patron fabrique abstraite. Pour que l'utilisateur accède aux objets de la BDD « Base » en utilisant les liens de BDD (Lien1, Lien2, Lien3).

On émettra la requête suivante sur NetBeans IDE 7.2.1 pour afficher les médecins et les infirmiers :

Select * from utilisateur.Fabrique ;

Et les résultats de la requête sont illustrés dans la figure suivante :

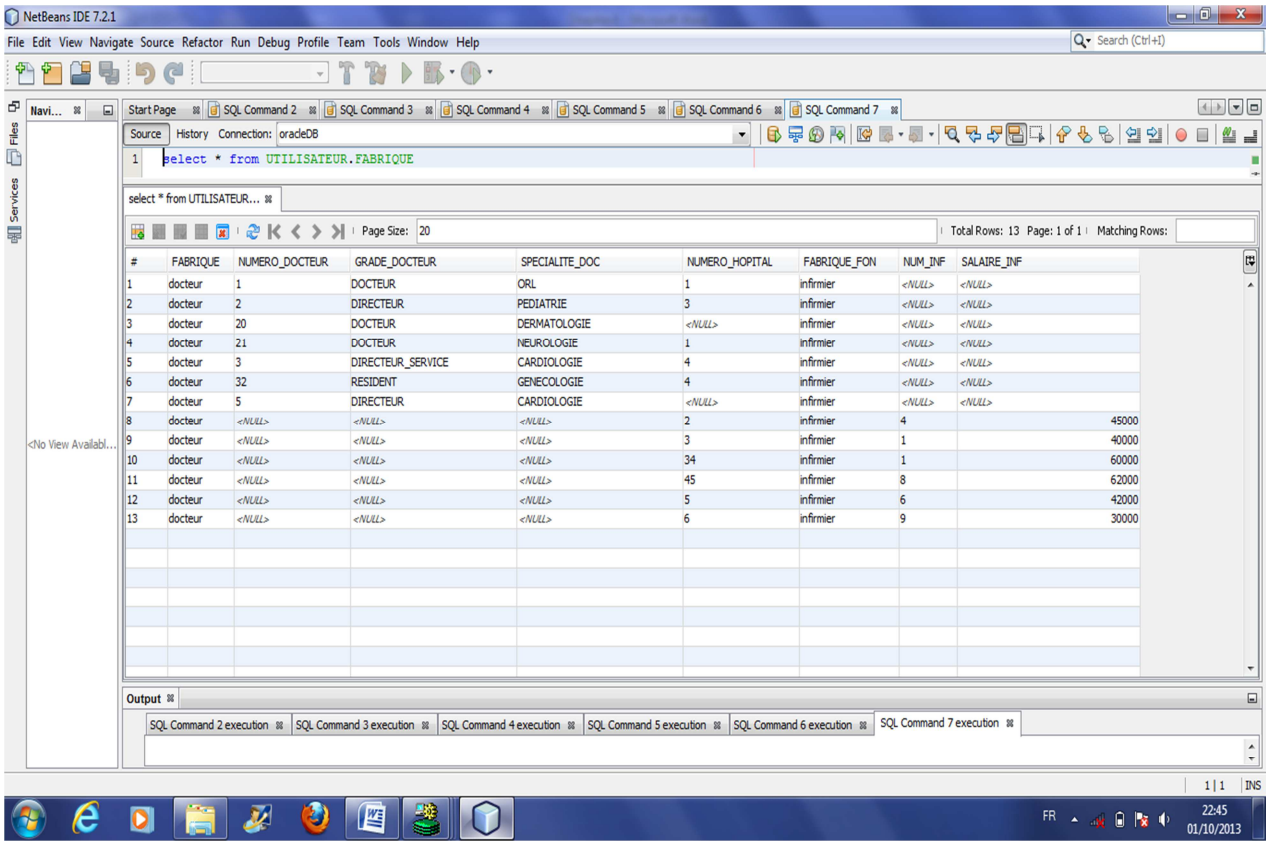

Figure IV.10 : Résultats d'application de patron fabrique abstraite

Conclusion :

Oracle 11g est un moyen de développement efficace pour montrer la possibilité d'intégrer plusieurs sous systèmes pour avoir un seul système homogène au moyen des liens de bases de données et les patrons de conception.

L'utilisation de liens de bases de données permet aux requêtes SQL de faciliter l'accès à des objets des autres bases de données que ce soit distantes ou bien locales situées sur une même machine d'où note cas, et l'utilisation des patrons de conception garantissent l'accès à des données des différents sous systèmes sans les consulter chacun à part et cela au moyen des interfaces communes tel que l'interface adapt, façade , pont et fabrique abstraite.

Conclusion générale

La généralisation d'une approche par objets de la conception de logiciel a permis de focaliser la tâche de conception sur la description des schémas (ou motifs) d'interactions entre objets, et d'assigner des responsabilités à des objets individuels de telle sorte que le système résultant de leur composition ait les propriétés extra-fonctionnelles voulues. L'aspect récurrent de certains de ces schémas d'interactions entre objets a conduit à vouloir les cataloguer sous la forme de *design patterns* ou patrons de conception.

Au cours de dernière décennie, on est ainsi passé progressivement d'une approche artisanale et approximative de la conception à une application rationnelle de solutions ayant fait l'objet d'un catalogage systématique parce que issues des meilleures pratiques du domaine. Aujourd'hui, on aborde un nouveau défi concernant l'automatisation de l'application de ces solutions de conception à l'aide des notions de transformation de modèle ou de tissage d'aspects. Le défi reste bien sûr toujours de concilier automatisation et flexibilité afin de ne point restreindre la créativité des concepteurs.

La procédure d’identification de patrons de conception n’est pas encore une tâche formelle. Plus d’une fois, nous avons identifié des variantes de patrons. La structure divergeait de celle proposée par le GoF, mais leur comportement possédait les mêmes caractéristiques que celles du livre. L’identification de patrons de conception, que nous avons effectuée au cours de ce travail, contribuera à l’établissement et la confirmation de règles à propos des métriques d’un logiciel ceci aurait pour but d’automatiser la tâche fastidieuse que nous avons dû effectuer manuellement. Ainsi, comme en psychanalyse, cerner un patron de conception est une tâche abstraite, qui requiert une appréciation humaine, afin d’appeler l’esprit des Patrons.

L’interopérabilité des SI met en exergue de difficiles problèmes d’homogénéité aussi bien au niveau de l’interaction homme machine (IHM) que du contenu fourni aux utilisateurs.

En effet, chaque SI intégré disposait préalablement de sa propre IHM. Les choix de conception, d’interaction et de design effectués dans un contexte d’usage spécifique ont conduit à des IHM hétérogènes.

Une nouvelle IHM est conçue, nécessitant un investissement lourd équivalent à la création de l’interface d’un nouveau système. A l’inverse l’intégration qui consiste à récupérer autant que possible les IHM préexistantes en les « adaptant » n’est pas sans difficulté. Le choix même des applications à intégrer peut alors être guidé par la capacité de celles-ci à voir leurs IHM être adaptées.

Le présent travail nous a permis d’acquérir des connaissances dans le domaine des bases de données, et de conforter nos connaissances en conception logicielle.

Enfin, on espère que notre travail sera d’une utilité à toute personne intéressée par ce sujet.

Bibliographie

[1] : Design Patterns**,** Outils pour la Gestion de Projets Axe ISI – 2007

[2] : Détection de particularités structurelles de modèles pour l'injection de patrons de conception Cédric BOUHOURS 2005 2006

[3] : www.Wikepedia.com

[4] : Mémoire Magister « Patron de conception (design pattern) pour une solution d'interopérabilité » M^{r}. kerbiche 2008

[5] : patrons de conception, Ada Diaconescu

[6] : http://www.blackwasp.co.uk/GofPatterns.aspx

[7] : Raffinement Incrémental de Modèles Evénementiels
Thierry Lecomte – ClearSy, Dominique Méry – Loria , Dominique Cansell – Loria, 2007.

[8] : Patrons de conception (design pattern), xavier crégut, Département Télécommunications & Réseaux (ENSEEIHT)

[9] : Les patrons de conception: Représentation et mise en œuvre, Ghizlane El Boussaidi Hafedh Mili, Laboratoire de recherche sur les Technologies du Commerce Electronique, Université du Québec à Montréal.

[10] : Les concepts de l'ingénierie et de l'intégration des systèmes, Systèmes et processus d'Ingénierie Systèmes, Yann Pollet, Conservatoire National des Arts et Métiers, Chaire d'intégration des systèmes

[11] : THESE Doctorat En Science, «Proposition d'une architecture d'intégration des applications d'entreprise basée sur l'interopérabilité sémantique de l'EbXML et la mobilité des agents » Présentée par : Razika DRIOUCHE, Dirigée par : Pr. M^{me} Zizette BOUFAIDA.

[12] : www.Techno-Science.net

[13] : http://fr.wikipedia.org/w/index.php?title=Interopérabilité_en_informatique

[14] : L'EDI : concepts et définitions Projet TZ67 Chef de projet : Richard EMEYRIAT
Équipe projet : Adrien ARRAIOLOS, Nicolas DE LA SELLE.

[15] : Mémoire master « Mise en place d'une plate-forme d'interopérabilité d'accès aux manuscrits arabes numérisés basée sur Dublin Core » dirigé par : Mr SOUALAH, réalisé par M^{elle} DJEFEL Ouzna, M^{elle} OUAGUENOUNI Nacera. Promotion 2011/2012.

[16] : www.Developpez.com

[17] : Architecture des applications, Cedric Dumoulin.

[18] : www .Commentçamarche.com

[**19] :** www.Oracle.com

[20] : Systèmes de gestion de bases de données réparties et mécanismes de répartition avec Oracle, Rim Moussa, université Carthage