République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud Mammeri de Tizi-Ouzou

Faculté de génie électrique et d'informatique

Département d'informatique

Mémoire de Fin d'études

En vue de l'obtention du diplôme Master en Informatique

Option : Conduite De Projets Informatique

Thème

Patrons De Conception Pour Une Solution D'intégration De Systèmes D'informations

Proposé et dirigé par :

M^{r} : KERBICHE M'hand

Réalisé par :

M^{r} : AZIZ Mohamed

M^{elle} : RABIA Siham

Promotion 2013 / 2014

Remerciements

Nos plus vifs remerciements vont à notre promoteur Mr KERBICHE Mohand pour nous avoir proposé ce sujet et nous avoir dirigés tout au long de sa réalisation. Sa compétence, ses critiques et son bon sens nous ont largement aidé à réalise0r ce travail et nous lui exprimons ici nous sincères reconnaissances.

Nous tenons aussi à exprimer toute notre gratitude aux membres de jury d'avoir bien voulu accepter de juger notre travail.

Nous tenons à remercier toute personne qui nous a aidé pour l'aboutissement de notre travail.

Dédicaces

A Mes très chers parents.

A mes frères et mes sœurs.

A tout (es) mes ami (es).

Siham&Mohamed

Sommaire

Chapitre II : Patrons de Conceptions

Chapitre III : Analyse et Conception

Tables des illustrations.

Introduction Générale

Introduction générale

Les systèmes d'information ont connu ces dernières années un bouleversement considérable, provoqué par l'apparition de nouveaux besoins, En effet, des besoins d'interopérabilité et d'intégration s'imposent pour les applications interopérables et qui doivent fonctionner sur les environnements repartis tels qu'internet.

Dans ce contexte, certaines sources de données ont besoin de s'intégrer avec les autres pour fournir une valeur ajoutée à l'utilisateur final, Néanmoins des problèmes d'interopérabilité se posent et cela vue l'hétérogénéité entre les modèles.
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.

Dire de systèmes qu'ils sont intégrés signifie qu'ils partagent quelque chose, et qu'ils sont disponibles en un emplacement, alors que l'interopérabilité signifie généralement qu'ils sont capables de partager électroniquement des informations.

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.

On s'intéresse dans ce sujet à l'implémentation et la mise en œuvre de certain patron (motifs) de conception afin de faire intégrer les différentes sources. Le mémoire est structuré en quatre chapitres :

- ✓ Dans le premier chapitre : « Intégration et Interopérabilité », nous présentons les concepts généraux d'intégration et d'interopérabilité y compris leurs principes.
- ✓ Le second chapitre : On s'intéresse aux patrons de conceptions.
- ✓ Le troisième chapitre : « Analyse et conception »nous présentons notre cas d'étude et la méthode suivie pour la résolution de problèmes récurrents en utilisant quelques patrons de conception comme solutions aux difficultés.
- ✓ Le quatrième chapitre : « Réalisation »consacré à la réalisation et l'implémentation de l'application.

Chapitre I :

Intégration et Interopérabilité

I.1. Introduction :

Les systèmes d'information (si) sont une brique essentielle à l'organisation de nos entreprises. Ils sont aujourd'hui construits à partir de l'agrégation de systèmes informatiques (si) qu'il convient de maintenir et faire évoluer avec agilité et sans entropie. Constituent généralement des sources de données autonomes et hétérogènes.

De ce fait, l'interopérabilité entre ces systèmes d'information est complexe puisque les applications doivent être adaptées pour pouvoir déterminer, pour chaque requête, les sources de données pertinentes, la syntaxe requise pour l'interrogation, la terminologie (concept) propre à la source, et pour pouvoir combiner les fragments de résultats issus de chaque source en vue de construire le résultat final. Ce processus d'adaptation peut être plus ou moins complexe : exploitation des résultats d'une source pour interroger une autre, élimination des redondances, etc.

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'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.

Dire de systèmes qu'ils sont intégrés signifie qu'ils partagent quelque chose, et qu'ils sont disponibles en un emplacement, alors que l'interopérabilité signifie généralement qu'ils sont capables de partager électroniquement des informations.

I.2. Intégration de système :

I.2.1. Qu'est-ce qu'un système?

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.

I.2.2. Qu'est-ce que l'intégration ?

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.

L'intégration est souvent considérée comme l'un des derniers challenges de l'informatique.

Il désigne l'action d'intégrer. De son étymologie latine "integrare", intégrer signifie "rendre entier". L'intégration est de ce fait l'acte d'incorporation d'éléments constitutifs par lequel on va rendre un ensemble complet, lui conférer les propriétés attendues, et dans notre cas des propriétés essentiellement liées à l'interopérabilité et à la cohérence des systèmes d'information.

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.

I.2.2.1. Intégration de données :

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 (ExtractTransform 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.

I.2.2.2. Intégration des applications :

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, 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 dont l'objectif est de permettre de rationaliser et fluidifier le système d'information afin de le rendre plus flexible et plus réactif.

I.2.2.3. Intégration des processus :

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 qui a des règles et une sécurité de données. Ces données ne sont plus simples mais des objets métier 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.

I.2.3. Intégration d'applications d'entreprise(EAI) :

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.

Avantages de l'EAI:

- **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 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.

Inconvénients de l'EAI :

- **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, ...).

I.2.3.1. Principe de fonctionnement de L'EAI :

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 (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.

I.2.3.2. Services de l'EAI :

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.

I.2.3.3. Types de projets d'intégration :

L'EAI intéresse toutes les entreprises qui ont une implantation multi-sites, des matériels différents, des bases de données multiples, des systèmes multi applicatifs. 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.

- **Projets EAI stratégiques :**

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 processus. 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.

- **Projets EAI tactiques :**

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.

I.3. L'interopérabilité :

I.3.1. Définition de l'interopérabilité :

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.

I.3.2. Interopérabilité en informatique :

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.

I.3.3. Interopérabilité partielle :

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.

I.3.4. Définition de compatibilité :

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.

- **Différence entre compatibilité et interopérabilité :**

L'interopérabilité est similaire dans l'objectif à la compatibilité, mais avec la Volonté de s'étendre à tous les systèmes qui proposent les mêmes fonctionnalités. Les normes, les formats ouverts et les schémas pivots d'échange de données sont Une base sur laquelle peut se construire les initiatives autour de l'interopérabilité. Tous les logiciels sont compatibles avec un format commun, plutôt que d'être Compatibles avec chacun des logiciels.

I.3.5. Les principes d'interopérabilité :

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.

- **Interfaçage 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).

I.3.6. Les niveaux d'interopérabilité :

On distingue 3 niveaux d'interopérabilité :

I.3.6.1. L'interopérabilité technique :

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 Implementation	ISO 2709 ISAD(G)	XML RDF Espace de nom URL
Protocols	WAIS FTP Z39.50	HTTP OAI-PMH SRU/SRW

Figure I.1:Niveaux d'interopérabilité technique.

I.3.6.2. L'interopérabilité sémantique:

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.

I.3.6.2.1. Les principes d'interopérabilité sémantique :

L'interopérabilité sémantique se base sur trois 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.

I.3.6.3. L'interopérabilité organisationnelle :

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.

I.3.6.3.1. Les niveaux d'interopérabilité organisationnelle :

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-entreprises.

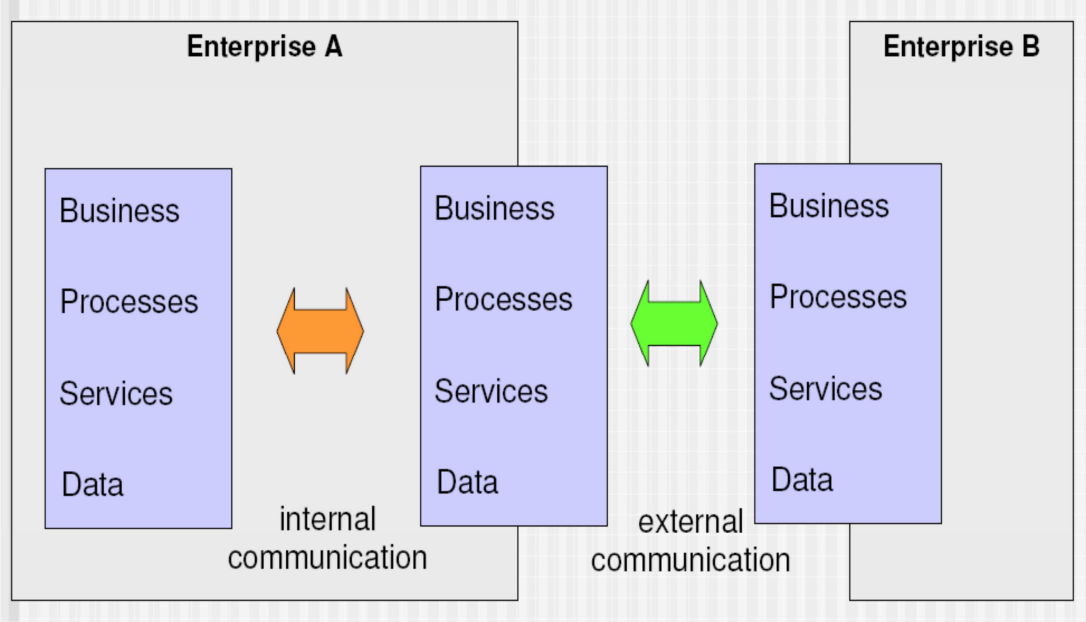


Figure I.2:Niveaux d'interopérabilité organisationnelle.

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.

I.3.7. Interopérabilité et échange de données informatisées(EDI) :

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.
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 ElectronicsEngineers) :
« 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. Il est clair que l'interopérabilité devient un des enjeux majeurs de l'échange de données informatisé.

I.3.8. Fonctionnement de l'EDI :

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 d'E1 standardisé par E2 ;
4 - La traduction du fichier standardisé au format propre à E2, par un traducteur spécifique à E2.

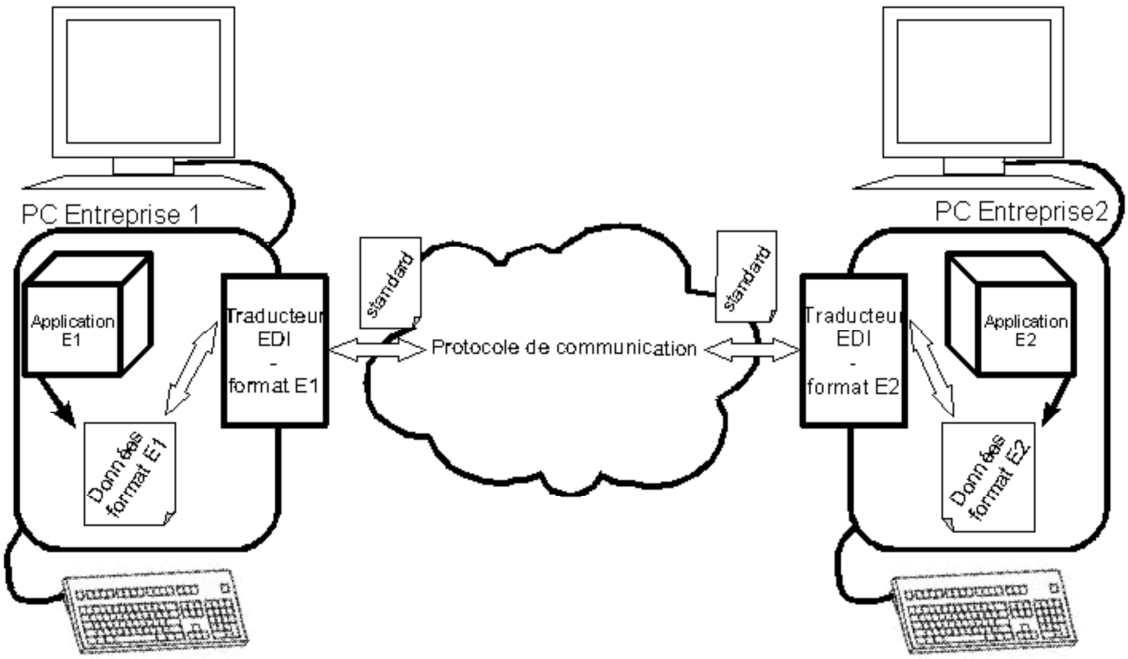


Figure I.3 : Fonctionnement de L'EDI.

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.

I.4. DEFINITIONS ELEMENTAIRE :

I.4.1. Définition d'une base de données :

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..

I.4.2. Définition d'une base de données répartie :

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.

I.4.2.1. Raisons de répartition des données :

- 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,
- Débit des disques.
- Pour améliorer le débit des E/Ss :
- Partitionnement des données,
- Accès parallèle aux données,

I.4.2.2. Buts de répartition des bases de données :

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.

I.4.3. SGBD (Système de Gestion de Bases de Données):

Un SGBD (en anglais DBMS, Data Base Management System) est un programme (logiciel) informatique de haut niveau qui permet de créer et de maintenir une base de données. C'est à dire :

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.

I.4.3.1.SGBD réparti :

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

I.4.4. Les liens de bases de données :

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 Delphi 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.

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 deuxième chapitre on s'intéressera aux patrons de conceptions.

Chapitre II :

Patrons de Conceptions

II.1. Introduction :

En génie logiciel, un patron de conception (*design pattern* en anglais) est un concept issu de la programmation orientée objet, destiné à résoudre les problèmes récurrents.

Ce sont des solutions standards pour répondre à des problèmes d'architecture et de design des logiciels. À la différence d'un algorithme qui s'attache à décrire d'une manière formelle comment résoudre un problème particulier, les design patterns, décrivent des procédés de conceptions généraux. On peut considérer un design pattern 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 dépendants 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 design pattern a donc une grande influence sur l'architecture logicielle d'un système.

. De façon générale, on utilise un design pattern pour diminuer le temps nécessaire au développement d'une application et pour augmenter la qualité du résultat attendu à un traitement donné.

II.2. Les patrons (Patterns):

II.2.1. Définition de patron (Pattern) :

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.

II.2.2. Catégories de Patterns :

- **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. (pipes, filters, brokers, blackboard, MVC, …)
- **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.

II.2.3. Framework :

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.

II.2.3.1. Techniques de modélisation :

Modélisation de patrons de conception :

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.

Modélisation des patterns architecturaux :

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 conséquent, 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 Framework.
- 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.

II.3. Patron de conception (Design pattern) :

II.3.1. Définition :

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.

II.3.2. Patron de conception prouvé :

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.

II.3.3. Pourquoi définir des patrons de conception ?

- 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

II.3.4. Choix d'un patron de conception :

- 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 !

II.3.5. 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.

II.3.6. Ce qu'il ne faut pas attendre des patrons de Conception :

- 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

II.3.7. Les problèmes d'utilisation et d'application des patrons :

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.

II.3.8. Description des patrons par GoF :

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.

II.3.9. Types de patrons de conception selon GoF :

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 II.1 : Catalogue GoF patterns.

Portée de Classe

- Focalisation sur les relations entre classes et leurs sous-classes
- Réutilisation par héritage

Portée d'Instance

- Focalisation sur les relations entre les objets
- Réutilisation par composition

II.3.9.1. Les patrons créateurs :

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.

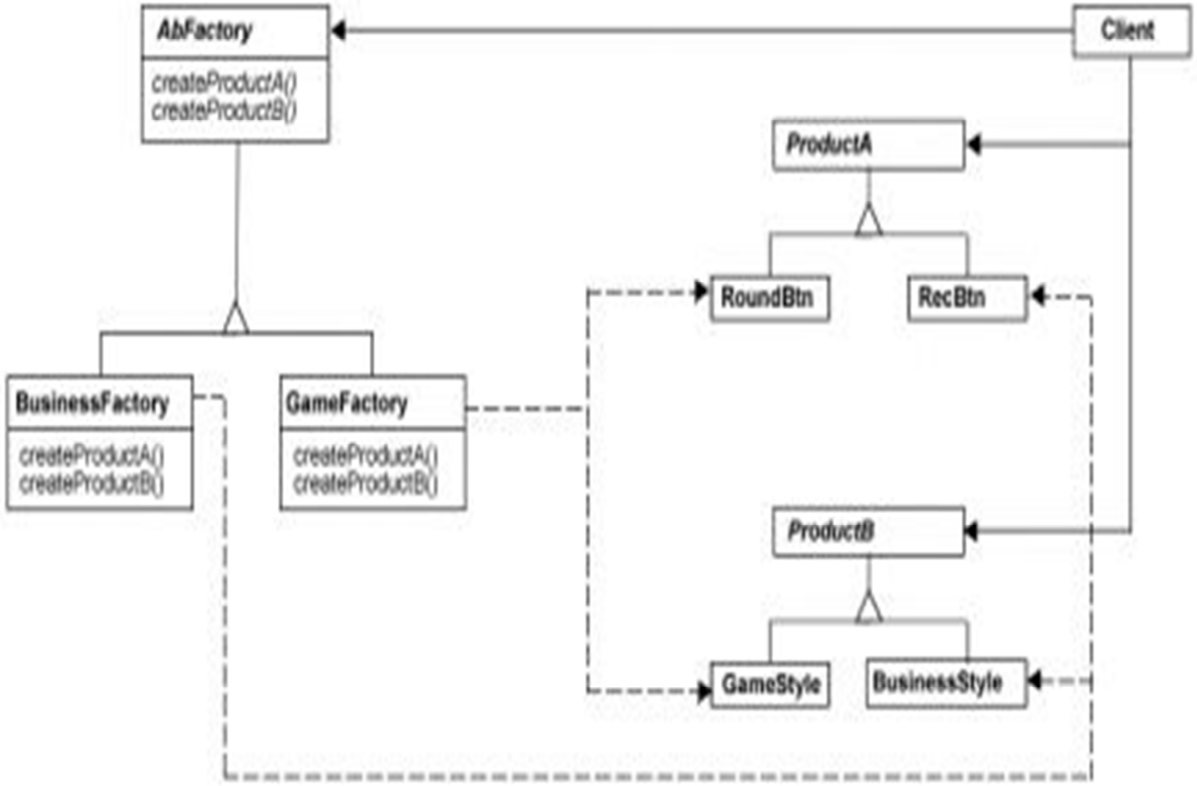


Figure II.2 : Patron de conception Fabrique abstraite.

On utilise le pattern Fabrique Abstraite :

- Quand on doit créer une famille d'objets qui se ressemblent.
- Quand les objets d'une famille peuvent évoluer.

➢ **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.

On utilise le pattern Monteur :

- Quand on doit créer un objet complexe qui nécessite plusieurs étapes.

- **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.

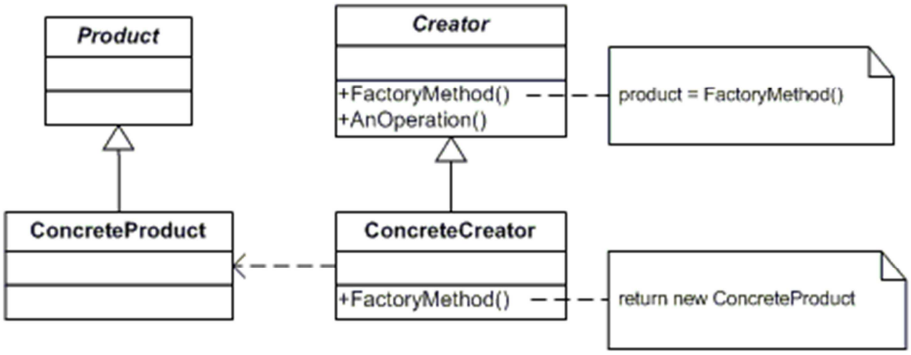


Figure II.3 : Patron de conception Fabrication.

On utilise le pattern Fabrique :

- Quand on doit créer un objet concret en fonction d'un paramètre donné.
- Quand on a besoin de faire évoluer ou ajouter des classes concretes sans modifier la classe abstraite.

- **Le Prototype :** En utilisant une instance type, il crée de nouveaux objets par clonage. On utilise le pattern Prototype :
  - Quand on doit créer de nouveaux objets à partir d'objets existants soit par copie, soit par clonage.
  - Quand la création d'un objet est plus coûteuse que de le copier.

- **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.

On utilise le pattern Singleton :

- Quand un objet ne doit posséder qu'une seule et unique instance sous peine de générer une erreur.
- Quand un objet créé deux fois est susceptible de générer une erreur.

II.3.9.2. Les patrons structurels :

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.

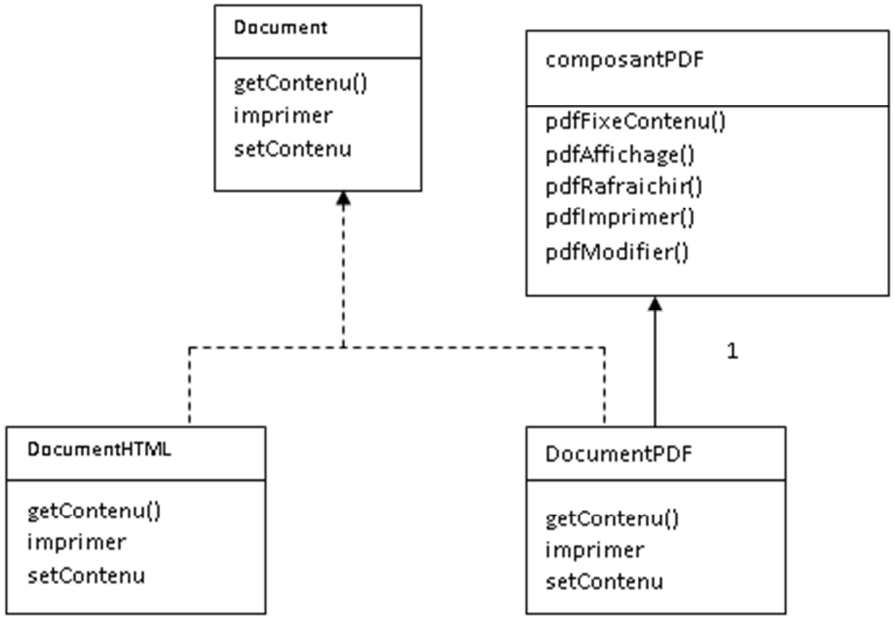


Figure II.4: Patron de conception adaptateur.

- **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.

II.3.9.3. Les patrons comportementaux :

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
    - o (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).
- Les patrons comportementaux classés selon Gof sont :
- Motifs de comportement d'objets avec l'utilisation de l'association entre objets :

- ➢ **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.

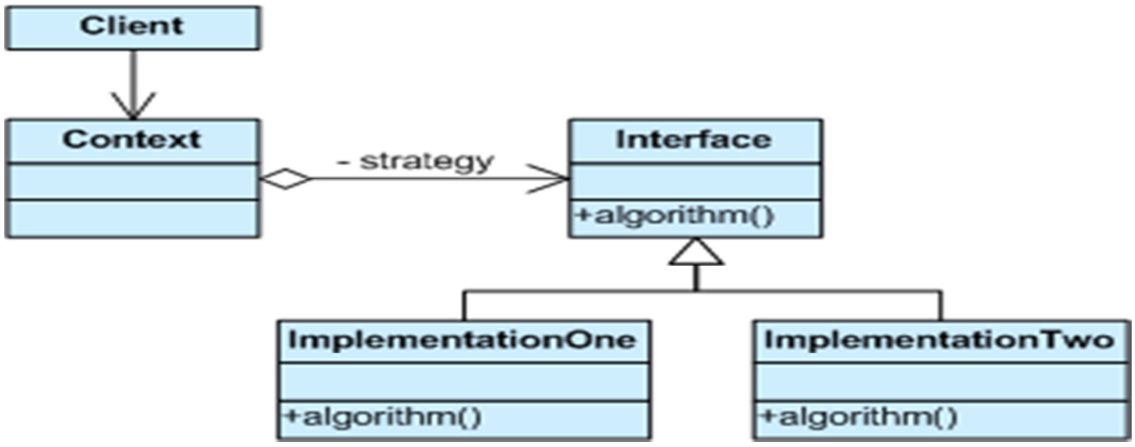


Figure II.5: patron de conception stratégie.

- **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 que nous avons défini selon l'ouvrage de gang of qui représente un espace très riche de composition ou de simplification du développement objet.

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

Chapitre III :

Analyse et Conception

III.1. 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 deuxième chapitre.

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

III.2. Le modèle MVC (Model-View-Controler) :

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 :

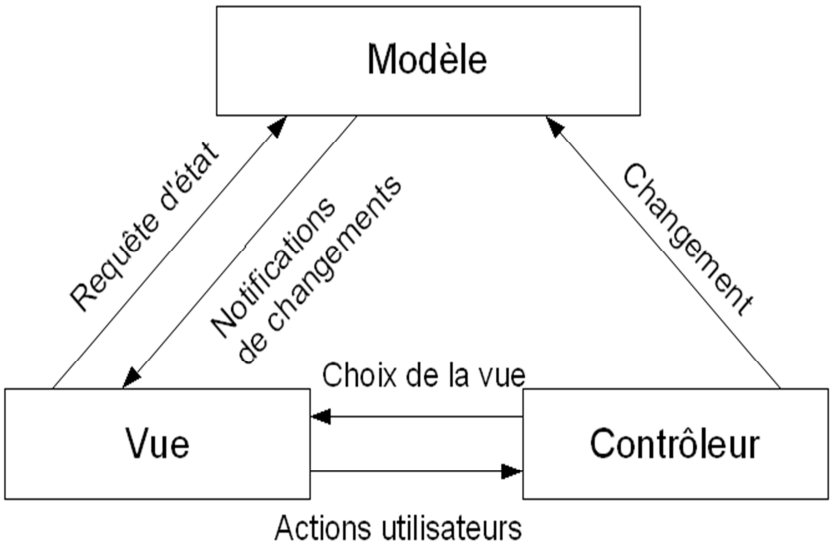


Figure III.1: Model MVC.

III.2.1. Développement sans respect du modèle MVC:

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.2.2. Développement avec respect du modèle MVC :

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.2.2.1 Avantages de MVC :

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

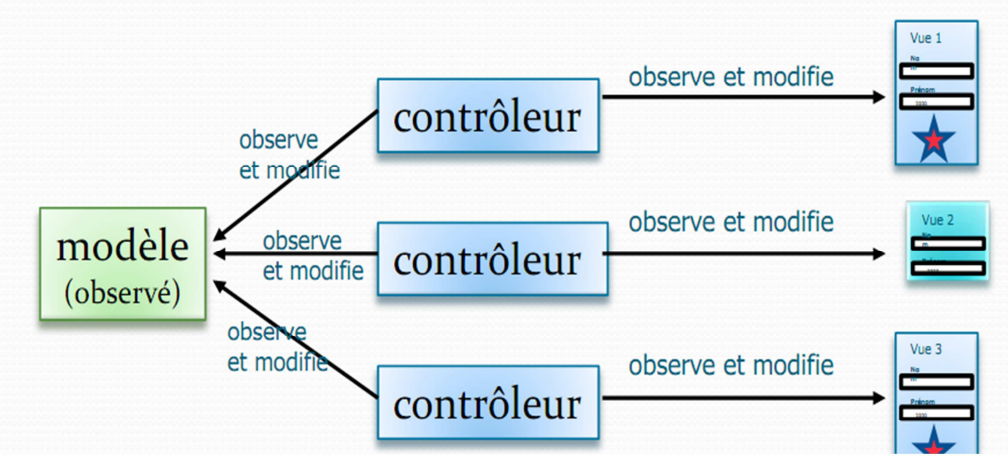


Figure III.2 : Fonctionnement de MVC.

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.3. Présentation de cas d'étude :

Considérons un système à développer qui est constitué de deux bases de données hospitalières **base1**, **base2** des deux villes : **ville1**, **ville2** 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).

III.4. Description de système actuel :

Comme le montre la figure 11 notre système contient deux bases de données (Base1, Base2) 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.

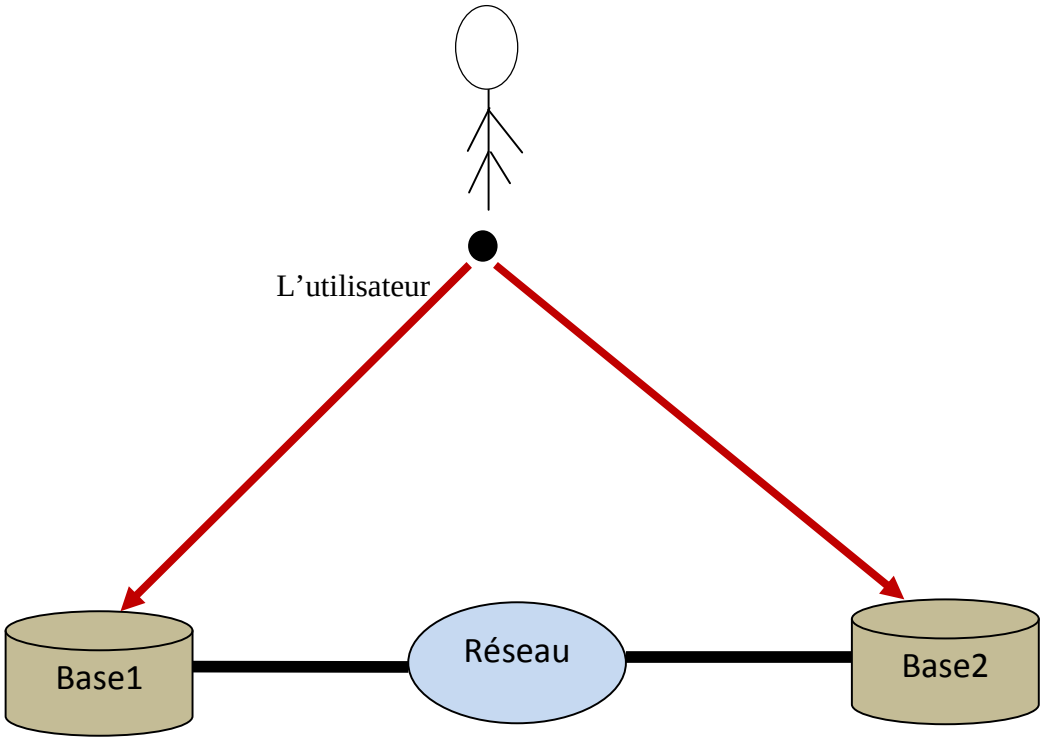


Figure III.3 : La structure globale du système actuel.

Le problème traité :

Comment assurer l'intégration des deux 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 deux bases (Base1, Base2).

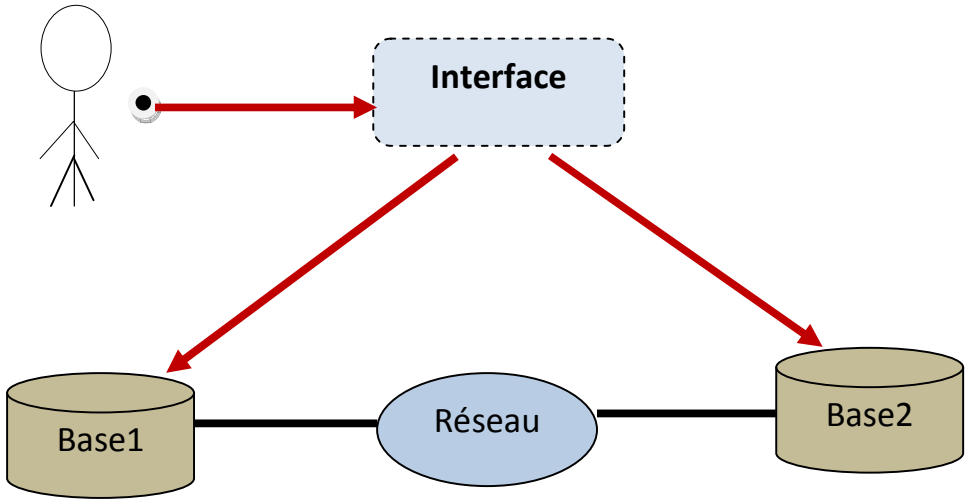


Figure III.4: La structure globale du système final.

III.5. 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.5.1. 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.

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 :

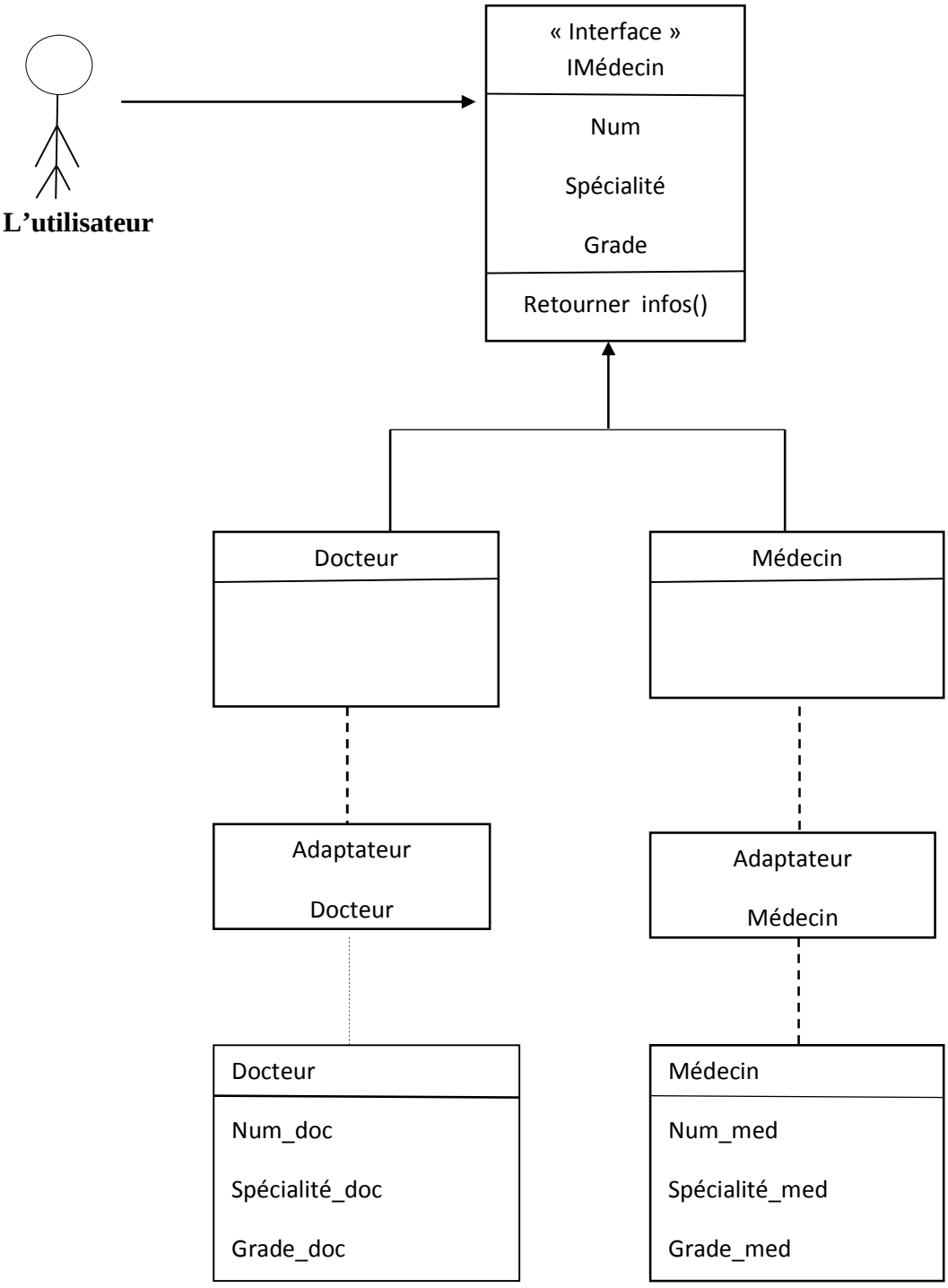


Figure III.5:Patron Adaptateur.

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 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 par l'état de l'exécution.

III.5.2. 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.

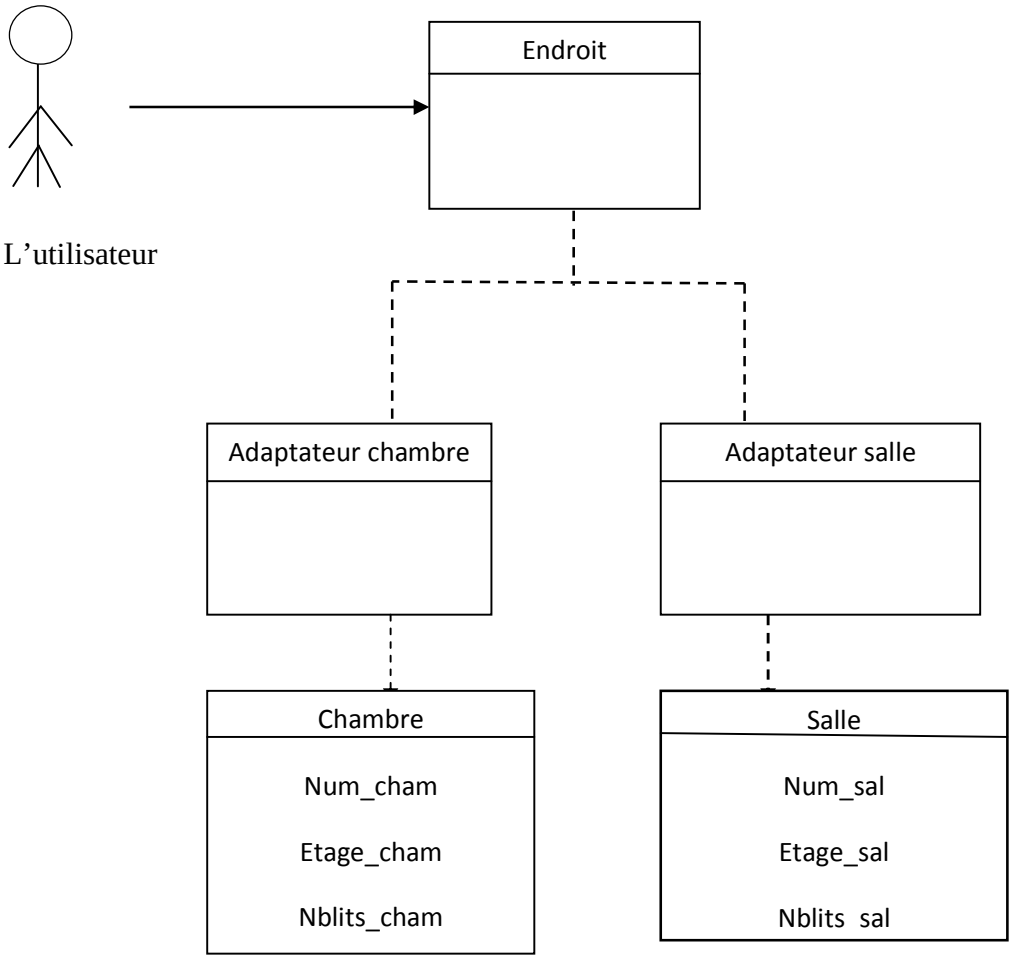


Figure III.6 : Utilisation de patron Façade pour connaitre les pièces hospitalières.

Dans notre cas, on à un système composé de deux 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.5.3. 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 **hospice** implique l'ajout de ces services **subdivision1** et **subdivision2** 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**.

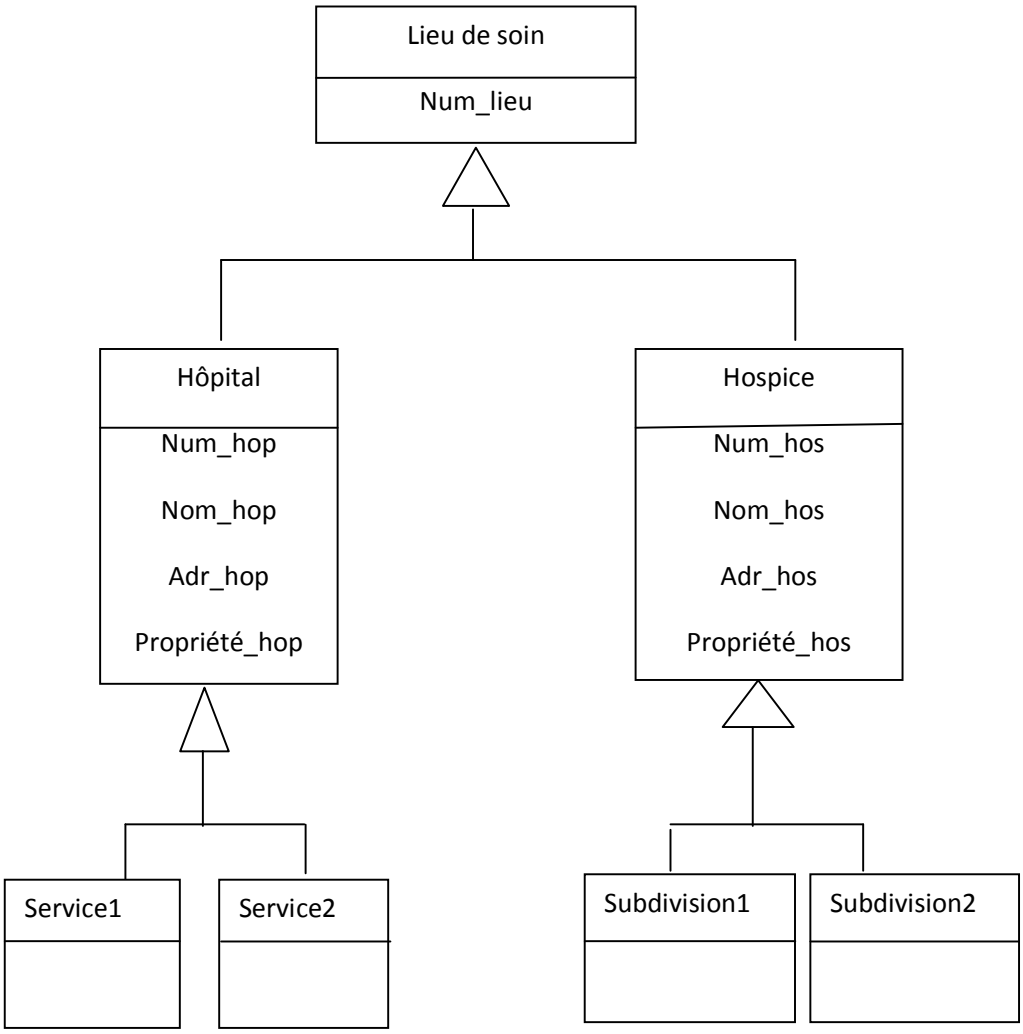


Figure III.7 : Structure globale du système avant application du pont.

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 **hospice** 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**.

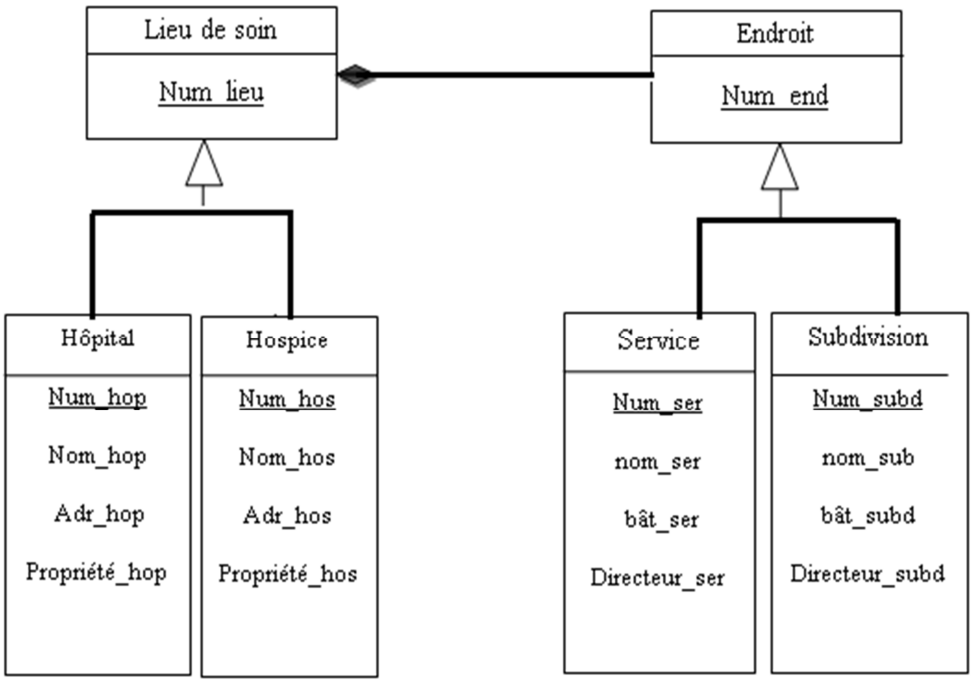


Figure III.8 : Structure globale du système après application du pont.

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.5.4. 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.

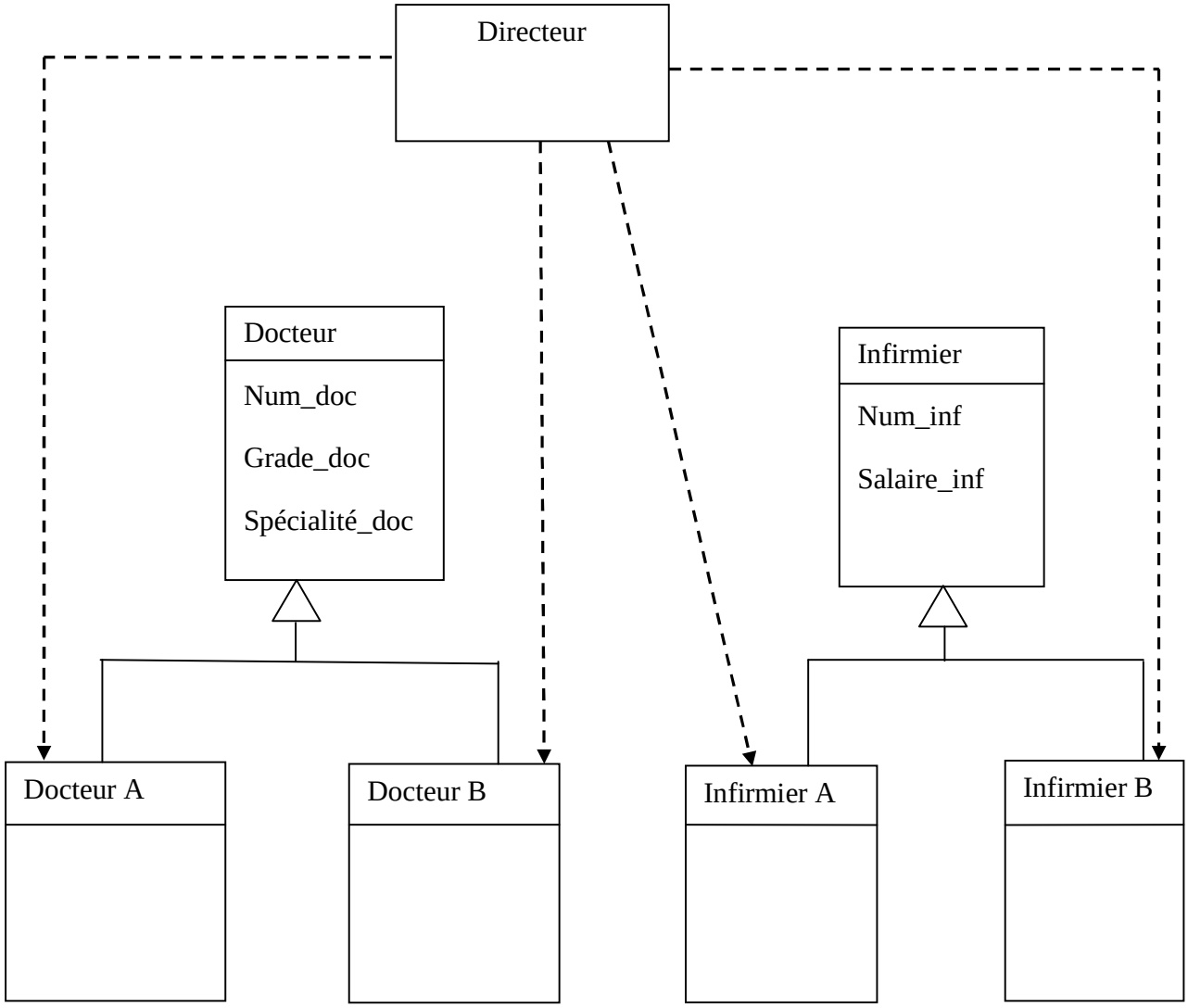


Figure III.9 : Structure du système avant application de la fabrique abstraite.

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 :

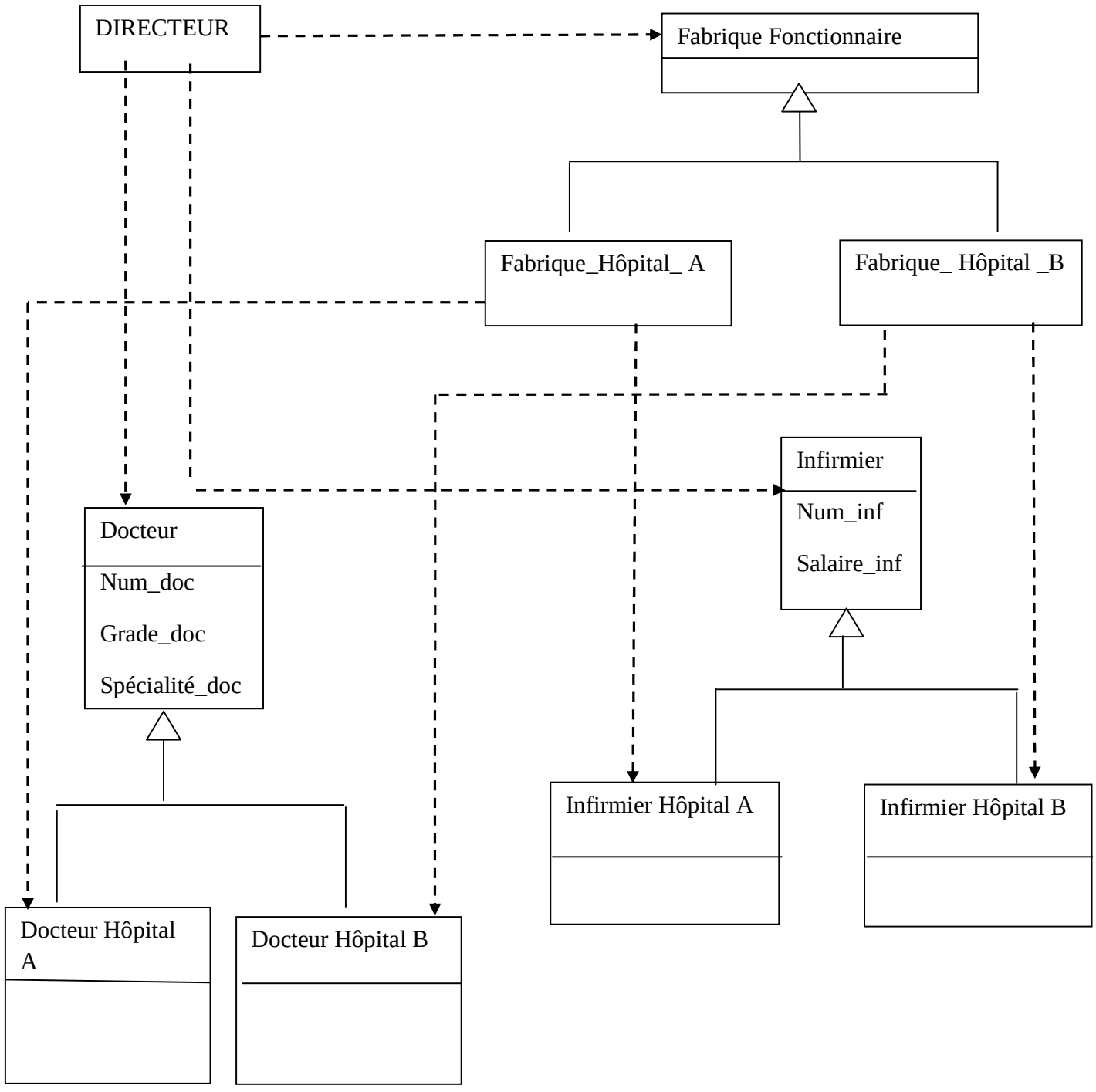


Figure III.10 : Création d'objets à l'aide de la fabrique abstraite.

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 FactoryMethod
- 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

IV.1. 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.2. Outils de développement :

IV.2.1.Delphi :

L'Embarcadero Delphi (souvent abrégé en Delphi) est à la fois un environnement de développement intégré (EDI) et un langage de programmation orienté objet.

Delphi dispose de nombreux composants permettant d'accéder aux bases de données et de les exploiter. Delphi répartit ces composants selon les mécanismes d'accès aux données qui diffèrent d'une technologie à l'autre. Sur la palette des composants, les composants bases de données sont regroupées dans six pages :

- La page BDE de la palette de composants contient les composants qui utilisent le moteur de base de données Borland (BDE, Borland DatabaseEngine). Le BDE définit une API importante pour l'interaction avec les bases de données. De tous les mécanismes d'accès aux données, le BDE gère le plus large éventail de fonctions et offre les meilleurs utilitaires. Il représente le meilleur support de manipulation des données dans les tables Paradox ou dBASE. Toutefois, c'est le mécanisme le plus compliqué à déployer. Pour plus d'informations sur l'utilisation des composants BDE, voir Utilisation du moteur de bases de données Borland.

- La page ADO de la palette de composants contient les composants qui utilisent ActiveX Data Objects (ADO) pour accéder aux informations de base de données via OLEDB. ADO est un standard Microsoft. De nombreux pilotes ADO permettent de se connecter à différents serveurs de bases de données. Grâce aux

composants ADO, vous pouvez intégrer votre application dans un environnement ADO (par exemple, en recourant à des serveurs d'applications ADO). Pour plus d'informations sur l'utilisation des composants ADO, voir Utilisation des composants ADO.

- La page dbExpress de la palette de composants contient les components qui utilisent dbExpress pour accéder aux informations de base de données. dbExpress est un ensemble de pilotes légers qui offrent l'accès le plus rapide aux informations de base de données. En outre, étant également disponibles sous Linux, les composants dbExpress autorisent le développement multi-plate-forme. Toutefois, les composants base de données dbExpress prennent en charge l'éventail le plus réduit de fonctions de manipulation de données. Pour plus d'informations sur l'utilisation des composants dbExpress, voir Utilisation des ensembles de données unidirectionnels.

- La page InterBase de la palette de composants contient les composants qui accèdent directement aux bases de données InterBase, sans passer par une couche motrice distincte. Pour plus d'informations sur l'utilisation des composants InterBase, voir Introduction à InterBase Express.

- La page AccèsBD de la palette de composants contient des composants pouvant être utilisés avec n'importe quel mécanisme d'accès aux données. Cette page comprend TClientDataset, qui peut utiliser les données stockées sur disque ou, par le biais du composant TDataSetProvider de cette même page, les composants d'un des autres groupes. Pour plus d'informations sur l'utilisation des ensembles de données client, voir Utilisation d'ensembles de données client. Pour plus

d'informations sur TDataSetProvider, voir Utilisation des composants fournisseur.

IV.2.2.Architecture interne « accès aux données de bd » :

Delphi met en place un certain nombre d'utilitaires et de mécanismes internes pour qu'une application ait accès aux données gérées par les différents SGBDR. Il fournit en particulier un moteur de base de données interne appelé BDE (Borland DatabaseEngine), qui permet de créer et de gérer des bases de données locales. C'est BDE qui assure par ailleurs la communication avec les autres bases de données.

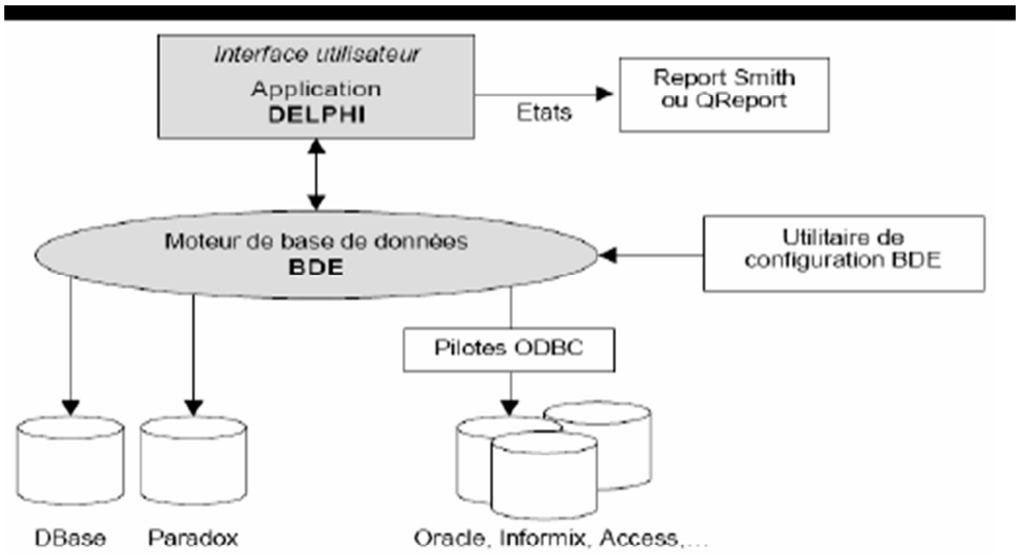


Figure IV.1: Architecture interne d'une BDD sous Delphi.

- Moteur BDE : Le moteur de base de données Borland BDE (BorlandDatabaseEngine) est l'élément central d'une application de gestion de base de données créée avec Delphi. Il est inclus directement dans les composants spécifiques fournis avec Delphi. Un programmeur n'a donc pas à s'en occuper et il n'apparaît pas dans l'arborescence de l'application créée. Par contre l'exécutable généré est plus important. L'application Delphi créée est essentiellement

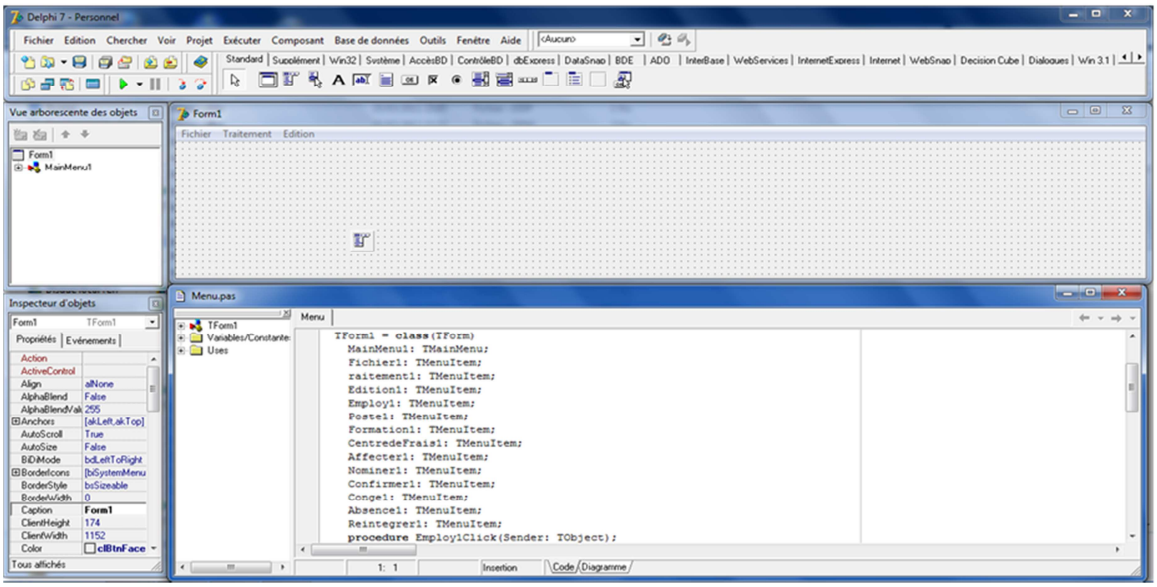
constituée d'une l'interface utilisateur permettant d'accéder de manière transparente à BDE.

- Module de Bases de Données :Les structures des différentes tables utilisées dans l'application sont mises au point par un utilitaire spécial : DBD(DataBase Desktop).Cet utilitaire, accessible par une icône spécifique ou par le menu Outils' permet de créer les différentes tables de l'application (dénominations, types et tailles des différents champs et définition des clés), de définir les différents index, de créer les liens entre les différentes tables.
- SGBDR (SGBD Relationnel) : Il est possible d'utiliser des tables déjà conçues par d'autres.
- Pilotes ODBC : Les pilotes ODBC permettent l'accès à différentes bases de données et serveurs SQL non reconnus directement par Delphi.

IV.2.3.L'interface de développement Delphi :

L'environnement de développement s'appuie sur un éditeur d'interface graphique associé à un éditeur de code source. Il doit son succès à sa facilité d'utilisation pour développer des applications graphiques et/ou liées aux bases de données.

A l'ouverture de DELPHI cinq fenêtres s'affichent :



FigureIV.2 : Page d'accueil de DELPHI.

IV.3.Interfaces de notre application :

Figure IV.3 : Page d'accueil de l'application de gestion HOPITAL.

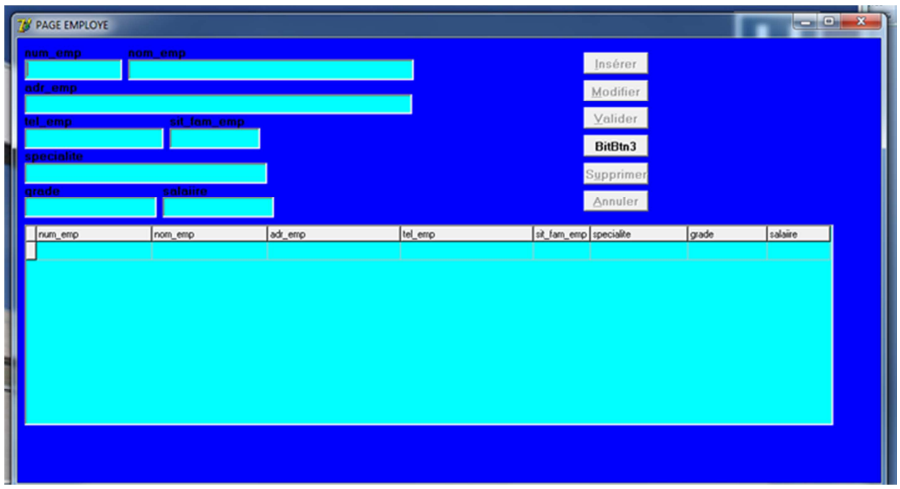


Figure IV.4 : Page employé de l'application de gestion HOPITAL

Figure IV.5 : Page d'accueil de l'application de gestion HOSPICE.

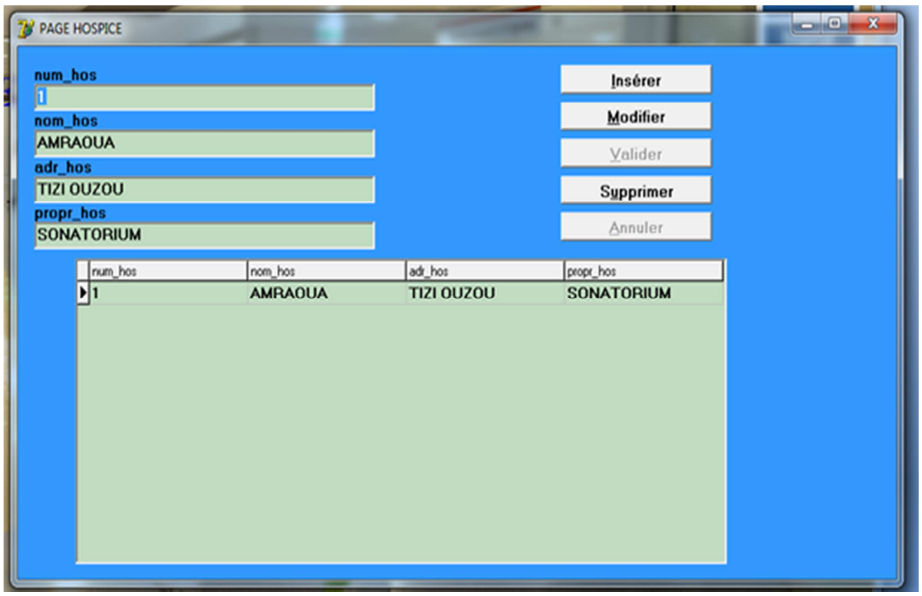


Figure IV.6: Page hospice de l'application de gestion HOSPICE.

Figure IV.7 : Page d'accueil de l'application d'intégration HOPITAL et HOSPICE.

Conclusion :

Les applications de bases de données permettent aux utilisateurs d'interagir avec les informations stockées dans les bases de données. Les bases de données permettent de structurer les informations et de les partager entre plusieurs applications.

DELPHI 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

Conclusion Générale

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. **Gerald Croes «**Atelier design pattern …»Tutorial.
2. **P. Laroque** «design pattern» cours Universit2 Cergy pontoise
3. **Sylvain Sherrier** «Les design pattern GoF» Tutoriel
4. Mathieu Musard «création de base de données» École supérieure d'informatique à paris
5. **DENIS Roegel** «oracle-sql» IUT Nancy.
6. **FABRICE JOUANOT** «un model sémantique pour l'interopérabilité de systèmesd'information» Université de Bourgogne.
7. **Yousra TEGMOUTI**Thèse fin d'études «modélisation et implémentation de designs pattern» Université de MONTREAL.
8. **DAN Vodislav** «architecture d'intégration de données » Université de Cergy pontoise.
9. **M.SINI**«cours sur les systèmes d'informations» master 2 cpi Ummto.
10. **RIM Moussa** «systèmes de gestion de base de donnes reparties et mécanismes de répartitions avec oracle «Université Carthage.

- www.developpez.com/comprendrelesdesignpattern
- www.developpez.com/introductionalaprogrammationorienteobjet
- www.wikipidia.com/interoperabilite
- www.wikipidia.com/integrationdesystemesdinformation
- www.google.fr/itroductionmvc
- www.bdd-ora.com
- www.commentcamarche.com/basededonnes
- www.sitedezero.fr