Résumé

Pour commencer notre travail, il nous a fallu faire une étude théorique sur tous les outils nécessaires pour le développement d'une application WEB et plus précisément les outils les plus récents et les plus utilisés. Ceci nous a amené d'approfondir nos connaissances sur les framework, qui est une technologie WEB très récentes, d'étudier plusieurs d'entre eux, de les tester et de les implémenter pour pouvoir en manipuler l'un deux qui est Hibernate.

Nous avons également étudié le modèle client/serveur ainsi que le modéle MVC qui est une architecture moderne de structuration.

Ensuite, pour modéliser notre application, on a utilisé le langage de modélisation UML qui nous a permis de décrire les plans d'élaboration et de construction de notre application.

Enfin, pour réaliser notre application plusieurs technologies nous ont été nécessaires, on citera le langage XHTML, le java, le langage de base de données SQL et le HQL qui est un langage spécifique au framework Hibernate.

Après le passage par les différentes étapes précédemment citées, l'application a abouti à un logiciel fonctionnel qui répond globalement aux attentes de l'entreprise mais qui peut également êtres amélioré et perfectionner pour apporter plus de rentabilité à l'entreprise.

L'application que nous avons développée nous a permit d'acquérir des connaissances dans le domaine de la programmation, la conception et la réalisation d'application web client/serveur.

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI TIZI-OUZOU
X●ΘΛ●ΣX *[://:V* ●X *[●Λ [●O*
FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

Mémoire de fin d'etudes

En vue de l'obtention du diplôme Master en Informatique

Option : Conduite de projets informatiques(CPI)

Thème :

Méthodes agiles, UML pour la conception d'une application web pour une société d'assurance

« AXA assurance »

Dirigé par :

M r A.DIB

Réalisé par :

M elle DJEBALI Lilia

M elle HOCINE Thileli

Promotion

2013 /2014

Remerciement

Nous remercions DIEU le tout puissant qui nous a donné la force, la volonté et le courage pour accomplir notre travail.

Nous tenons à exprimer nos vifs remerciements accompagnés de notre gratitude tout d'abord à notre promoteur Mr A.DIB, pour avoir accepté de nous encadrer.

Pour nous avoir donné l'opportunité de travailler sur ce projet, pour son grand soutien scientifique et moral, pour les conseils, les suggestions et les encouragements qu'il nous a apportés et pour sa totale disponibilité durant le long de notre projet.

Nous tenons également à remercier les membres de jury de nous faire l'honneur de juger notre travail.

En fin nous exprimons notre profonde reconnaissance à toutes les personnes qui ont contribué de prés ou de loin pour le bon déroulement de ce travail.

En particulier nos chères familles et nos amis (es).

Dédicaces

Je dédie ce travail aux êtres qui me sont les plus chers au monde, ma chère mère et mon père à qui je dois mon existence et mes succés, Que Dieu le tout puissant les protège.

A mes chers frères et mes sœurs,

A ma chère nièce Lydia, ma sœur son mari et sa belle famille.

A toute ma famille.

A toutes mes amis

Et a tous ceux qui m'aiment.

Avec l'expression de tous mes sentiments de respect,

Je Dédie ce travail.

Lilia

Dédicaces

Je dédie ce travail aux êtres qui me sont les plus chers au monde, ma chère mère et mon père à qui je dois mon existence et mes succès, Que Dieu le tout puissant les protège.

A mon cher mari et à toute sa famille,

A mes chers frères et mes sœurs

A toute ma famille.

A toutes mes amis

Et a tous ceux qui m'aiment.

Avec l'expression de tous mes sentiments de respect,

Je Dédie ce travail.

Thileli

Sommaire

Sommaire

Chapitre III : Analyse et conception

Chapitre IV : La Réalisation

Liste des figures

Liste des figures

Introduction générale

Introduction Générale

Depuis l'apparition de l'informatique, l'être humain a toujours essayé d'exploiter cette science pour automatiser ses tâches quotidiennes de gestion, de communication, d'enseignement…

Il est aujourd'hui parfaitement admis que l'information constitue une ressource vitale pour les entreprises. La qualité des services informatiques a donc un impact certain sur la performance de ces dernières, à tel point qu'il est impensable, de ne pas disposer de cet outil.

On peut dire que l'informatique vient nous apporter de multiples conforts à notre mode de vie mais, au delà de l'utilisation individuelle de celle-ci c'est surtout la mise en communication des ordinateurs, qui a permis de révolutionner les méthodes de travail. Ce nouveau progrès offre aux utilisateurs de nouveaux outils de travail et leur permet d'améliorer leur rentabilité et leur productivité.

Au cours de ces derniers siècles, les réseaux d'entreprises ont connus une grande croissance. Ainsi, on assiste à une utilisation accrue d'applications sur le Web permettant d'accéder à l'information à tout moment et n'importe où, avec une connexion à l'Internet. Cela est rendu possible grâce aux progrès technologiques dans le domaine de l'informatique .Depuis des décennies, les projets informatiques sont gérés avec une approche classique basée sur des activités séquentielles: on recueille les besoins, on définit le produit, on le développe puis on le teste avant de le livrer au client.

« Quand on trouve une recette qui marche bien, on a du mal à la quitter même si l'on constate que son efficacité semble diminuer ; il existe une inertie due à la peur du changement, à la recherche de facilité ou à l'ivresse du succès (ce qui marchait hier doit marcher demain…).

C'est dans ce cadre que s'inscrit notre projet de fin d'études qui consiste à présenter méthodes agiles, UML pour la conception d'une application web pour une société d'assurance AXA.

Pour ce faire, on a adopté la structure suivante :

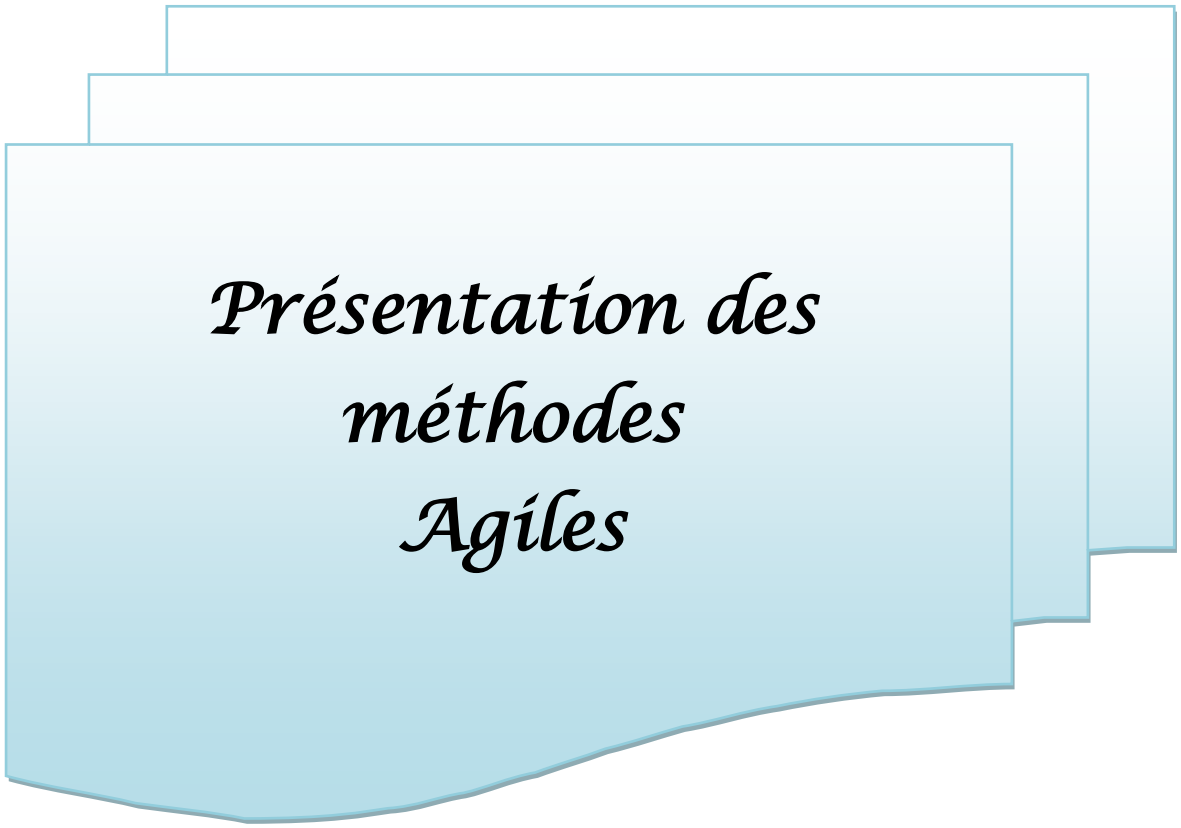
- **Chapitre I :** « Présentation des méthodes agiles» comporte la présentation des différentes méthodes agiles.

- **Chapitre II** : s'intitule « Généralités sur les assurances» comprend les généralités sur les assurances en particulier le groupe AXA qui est l'objectif de notre application.
- **Chapitre III :** «Analyse et conception» est consacré à l'analyse et à la conception de l'application.

- **Chapitre IV :** « La réalisation » dans lequel on décrit l'environnement de développement et l'implémentation de l'application.

Enfin, nous terminerons ce projet par une conclusion générale.

CHAPITRE I:



Présentation des méthodes Agiles

I. Introduction :

Depuis des décennies, les projets informatiques sont gérés avec une approche classique, le plus fréquemment « en cascade » ou son adaptation « en V », basée sur des activités séquentielles: on recueille les besoins, on définit le produit, on le développe puis on le teste avant de le livrer au client.

Ces méthodologies se caractérisent par un attachement farouche à tout planifier, « tout doit être prévisible », en tout début de projet. Voilà pourquoi on les qualifie d'approches «prédictives ». Un plan de management du projet décrit comment et quand le travail sera réalisé, les modalités de planification, d'exécution, de suivi et de clôture du projet.

Cette volonté persistante de vouloir piloter le projet par les plans (plan-driven development) a conduit les acteurs d'un projet à redouter, voire à s'opposer systématiquement à tout changement : changement dans le contenu ou le périmètre du projet, dans le processus de développement, au sein de l'équipe, bref à toute modification des plans initiaux, auxquels on doit rester conforme.

« Quand on trouve une recette qui marche bien, on a du mal à la quitter même si l'on constate que son efficacité semble diminuer ; il existe une inertie due à la peur du changement, à la recherche de facilité ou à l'ivresse du succès (ce qui marchait hier doit marcher demain…). Eh bien non !!!1 »

Fort du constat que les plans initiaux sont finalement toujours modifiés et que les besoins évoluent en permanence pour répondre aux changements du marché, ces approches prédictives se sont révélées trop « rigides » parfois, exposant les organisations à trop peu de réactivité dans le contexte de nouveaux projets stratégiques. Sont alors apparues, dans les années 1990, des méthodes moins prédictives, plus souples face aux besoins d'adaptation, facilitant ainsi l'agilité des organisations face aux contraintes du marché. Ce sont les méthodes dites « agiles».

II. Historique : [1]

Sans rentrer dans les détails, la première chose à savoir est que l'approche Agile n'est pas un effet de mode né de la dernière pluie. En effet la première approche de gestion de projet de développement itératif date de 1986. La première mise en œuvre de la méthode Scrum (la méthode Agile la plus utilisée, documentée et éprouvée aujourd'hui) date de 1993.

La seconde concerne un événement majeur rassemblant, en 2001, dix sept figures éminentes du développement logiciel pour débattre du thème unificateur de leurs méthodes respectives. De cet événement est né le Manifeste Agile rassemblant à la lueur des expériences de chacun les critères pour définir une nouvelle façon de développer des logiciels. Le terme « Agile » pour qualifier ce type de méthode est également né à cette occasion.

Aujourd'hui ces méthodes ont fait leurs preuves. Tout le monde (dans le monde de l'informatique) ou presque a au moins entendu parler d'une méthode Agile (Scrum, eXtreme Programming, RAD, Chrystal Clear,…). L'outillage associé est maintenant disponible sur le marché y compris dans le secteur Open Source. Les formations, certifications, conférences, livres, blogs se sont multipliés. Nous pouvons également noter la prise de position en faveur de l'approche Agile de la part des organismes faisant « autorité » en matière de gestion de projet informatique.

Origine des méthodes agiles : [1]

Le mouvement des méthodes agiles est né en 2001 aux États-Unis. Devant l'observation faite du taux important d'échec des projets, notamment dans les années 1990, dix-sept experts en développement logiciel, qui avaient chacun déjà mis au point et expérimenté de nouvelles méthodes, se sont réunis a la fin d'échanger et de trouver un socle commun de valeurs et de bonnes pratiques. Plus récemment, en 2005, a été publiée la « Déclaration d'interdépendance » avec, parmi ses signataires, des acteurs majeurs de l'Agile Alliance. Cette communauté a formalisé les valeurs mises en pratique qui les amenaient à rencontrer tant de succès et de bons résultats dans leurs projets.

III. Définition : [2]

Une méthode agile est une méthode de développement informatique permettant de concevoir des logiciels en impliquant au maximum l'utilisateur, avec une grande réactivité à ces commandes .c'est aussi une approche itérative et incrémentale, qui est menée dans un esprit collaboratif, avec juste ce qu'il faut de formalisme, ainsi elle génère un produit de haute qualité tout en prenant en compte l'évolution des besoins des clients.

Les méthodes agiles sont « adaptatives » plutôt que « prédictives », dans tous les projets, les exigences changent au cours du temps et le contexte évolue aussi. Pour cette cause les méthodes agiles se proposent de réserver un accueil favorable à ces changements, ce sont des méthodes itératives à planification souple qui leur permettent de s'adapter à la fois aux changements de contexte et de spécification du projet.

Pour une entreprise ou une organisation, l'agilité peut être définie comme l'aptitude à faire évoluer de façon continue ces structures et ses ressources ; en ce sens l'agilité est une préoccupation aussi ancienne que l'entreprise elle-même. Toutefois, cet objectif stratégique a surtout été poursuivi par le moyen de la qualité, qui vise à améliorer les processus pour innover ou améliorer ses produits et services.

Dans le domaine de l'informatique, l'approche par les processus est rapidement devenue synonyme de planification lourde de développement ; or l'accélération de la course à la rentabilité et la banalisation progressives des technologies de l'information créent, aujourd'hui, un contexte peut propice à ces méthodes traditionnelles.

C'est ainsi qu'est apparu à la fin des années quatre-vingt-dix, un mouvement de remise en cause des méthodes traditionnelles, perçues comme « bureaucratiques », soutenu par un management de plus en plus pressé.

IV. Le manifeste agile de logiciels : [2]

En février 2001, dix sept personnes se sont réunis et sont tombés d'accord que durant leurs expériences de développement, la réussite des projets et le succès se produisait en utilisant un ensemble de valeurs et de pratiques totalement différentes des approches s'appuyant sur des plans ou de la documentation.

De cette réunion résultat une déclaration innovante présentant l'approche « développement logiciel agile ».le document qu'ils mirent en œuvre et signèrent, dénommé « manifeste pour le développement logiciel agile »résume leurs convictions et met en avant les valeurs suivantes : (entre parenthèses, les citations du manifeste).

- **L'équipe** (« Les individus et leurs interactions, plus que les processus et les outils ») : dans l'optique agile, l'équipe est bien plus importante que les outils (structurants ou de contrôle) ou les procédures de fonctionnement. Il est préférable d'avoir une équipe soudée et qui communique, composée de développeurs (éventuellement à niveaux variables), plutôt qu'une équipe composée d'experts fonctionnant chacun de manière isolée. La communication est une notion fondamentale.

- **L'application** (« Des logiciels opérationnels, plus qu'une documentation exhaustive ») : il est vital que l'application fonctionne. Le reste, et notamment la documentation technique, est une aide précieuse mais non un but en soi. Une documentation précise est utile comme moyen de communication. La documentation représente une charge de travail importante, mais peut pourtant être néfaste si elle n'est pas à jour. Il est préférable de commenter abondamment le code lui-même, et surtout de transférer les compétences au sein de l'équipe (on en revient à l'importance de la communication).

- **La collaboration** (« La collaboration avec les clients, plus que la négociation contractuelle ») : le client doit être impliqué dans le développement. On ne peut se contenter de négocier un contrat au début du projet, puis de négliger les demandes du client. Le client doit collaborer avec l'équipe et fournir un compte rendu continu sur l'adéquation du logiciel avec ses attentes.

- **L'acceptation du changement** (« L'adaptation au changement, plus que le suivi d'un plan ») : la planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet. Les premières livraisons du logiciel vont souvent provoquer des demandes d'évolution.

Ces 4 valeurs se déclinent en 12 principes généraux communs à toutes les méthodes agiles:

1. La plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à forte valeur ajoutée.
2. Le changement est accepté, même tardivement dans le développement, car les processus agiles exploitent le changement comme avantage compétitif pour le client.
3. La livraison s'applique à une application fonctionnelle, toutes les deux semaines à deux mois, avec une préférence pour la période la plus courte.
4. Le métier et les développeurs doivent collaborer régulièrement et de préférence quotidiennement au projet.
5. Le projet doit impliquer des personnes motivées. Donnez-leur l'environnement et le soutien dont elles ont besoin et faites leur confiance quant au respect des objectifs.
6. La méthode la plus efficace de transmettre l'information est une conversation en face à face.
7. L'unité de mesure de la progression du projet est un logiciel fonctionnel (ce qui exclut de comptabiliser les fonctions non formellement achevées).
8. Les processus agiles promeuvent un rythme de développement soutenable (afin d'éviter la non qualité découlant de la fatigue).
9. Les processus agiles recommandent une attention continue à l'excellence technique et à la qualité de la conception.
10. La simplicité et l'art de minimiser les tâches parasites, sont appliqués comme principes essentiels.
11. Les équipes s'auto-organisent afin de faire émerger les meilleures architectures, spécifications et conceptions.
12. À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son processus de travail en conséquence.

Une méthode qualifiée d'agile doit donc se composer d'un ensemble de pratiques instrumentant le cadre décrit par les 12 principes généraux agiles et en conséquence s'inscrire dans le respect des 4 valeurs fondamentales ayant inspiré le Manifeste agile.

V. Les apports de l'agilité en informatique :

Les méthodes agiles sont, souvent, définies en réaction aux méthodes traditionnelles ; cette définition de nature « défensive » selon *Mr Philppe Michelin*, PDG de la BFD SA, cache leurs apports réels, notamment :

- L'utilisation d'un processus adaptatif, à la fois itératif et incrémental.

- Le contact rapproché avec l'expertise métier, par le travail en binôme (métier, informaticien) et la présence simultanée des membres d'équipes de développeurs sur le plateau pour favoriser la communication et l'interaction qui sont des valeurs essentielles dans les méthodes agiles.

- L'orientation vers l'humain avec un modèle de gestion avec délégation où les acteurs décident eux-mêmes comment travailler.

- La remise au premier plan de la qualité technique, par le travail permanent du code, permet d'obtenir de façon éprouvée, un logiciel maintenable.

- L'explicitation des exigences en continu et la présence active de l'utilisateur, qui minimise les risques liés à la gestion des évolutions.

- L'accent est mis sur les tests : écriture des cas de test en même temps avec l'écriture du code (c'est-à-dire l'ensemble des instructions exécutables par la machine).

Le processus lui-même est un autre aspect important de l'agilité, un processus qui change au fur et à mesure que l'équipe découvre ce qui fonctionne pour elle et modifie le processus en conséquence : dans ce type de démarche, l'accent est mis sur la prise en charge du changement qui reste bienvenu dans le développement.

VI. Les principales méthodes agiles : [3]

Les principes méthodes agiles se nourrissent toutes des valeurs et principe du Manifeste. Cependant, si elles ont un tronc commun de pratique, elles se différencient par leur degré de formalisme, le poids de la méthodologie dans la documentation produite, les étapes formelles, les revues, le rythme de projet ou le nombre de la longueur des itérations.

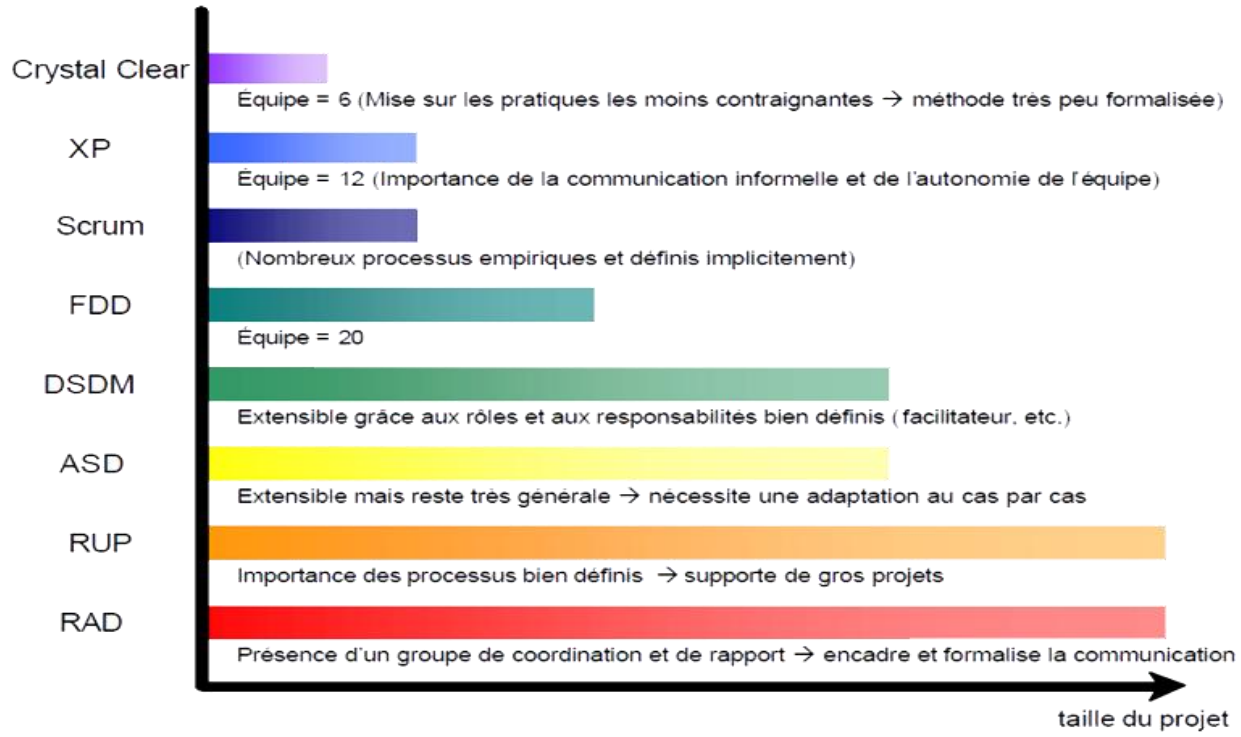


Figure I.1. : Les méthodes agiles les plus répondues.

VI.1. La méthode xp (eXtreme Programming) :

1 .Création : [4]

L'eXtreme Programming est une initiative de Kent Beck et Ron Jeffries, issue d'une, étroite collaboration avec Ward Cunningham, expérimentée en 1996 sur un projet pilote chez Chrysler.

2. Définition :

- XP C'est une méthodologie destinée à des équipes de taille réduite qui développent des logiciels dans un contexte où les besoins sont mal connus ou susceptibles d'évoluer rapidement. C'est une méthode qui fait partie de la famille innovante des processus agiles.
- XP se démarque des approches traditionnelles en mettant l'accent sur l'implication du client, le travail d'équipe, l'adaptation au changement et la robustesse des applications développées.

3. Les valeurs de XP : [5]

- **Communication** : l'effort de communication entre les différents intervenants est indispensable pour atteindre l'objectif commun. On privilégie la communication directe, dans le recueil et la clarification des besoins, dans la planification des itérations, dans la répartition et l'exécution des travaux. On évite ainsi bien des problèmes ou des situations de blocage.

- **Simplicité** : la solution la plus simple est la meilleure pour atteindre les objectifs ; grâce à cette simplicité, l'application pourra évoluer facilement, si nécessaire. La simplicité est applicable au client dans la définition de ces besoins, dans le choix des outils et du processus.

- **Feedback** : le retour d'information est essentiel pour valider le fait que le projet est sur la bonne voie : tests unitaires pour valider le fonctionnement du code, intégration continue pour détecter des anomalies, tests fonctionnels pour valider la conformité aux besoins, livraisons fréquentes…, autant de pratiques qui rendent plus aisées les adaptations éventuelles, sans attendre le terme du projet.

- **Courage** : du courage est absolument nécessaire pour adopter une démarche XP, démarrer un projet sans avoir toutes les spécifications, pour modifier du code existant sans en être l'auteur, et vice versa, pour appliquer les principes de communication, de *pair-programming*…

- **Du courage… et du respect** !

Les pratiques caractéristiques d'XP sont décrites ci-contre .On peut décrire ces pratique en les classant en trois grands groupes1 : activités de programmation, fonctionnement interne de l'équipe et gestion du projet.

4. Les pratiques relatives à la programmation :

- ✓ **Conception simple** (*simple design)* :

La solution la plus simple (concision, modularité, lisibilité) doit être implémentée. Plus l'application est simple, plus il sera facile de la faire évoluer dans les itérations suivantes. Il en va de même de la documentation.

- ✓ **Développement piloté par les tests unitaires** (*unit tests, test-first programming*) :

Avant d'implémenter une fonctionnalité, chaque développeur écrit un scénario de test unitaire pour le code qu'il produit ; il se constitue, ainsi, une batterie de tests qui sera réutilisée avant l'introduction d'un changement dans l'application.

- ✓ **Tests de recette** (*acceptance tests, customer tests*) :

En participant à la rédaction des procédures de tests de recette, le client précise ses besoins ; à la fin d'une itération, tous les tests sont lancés ; on mesure ainsi l'avancement du projet et on détecte les régressions éventuelles.

- ✓ **Remaniement continu** ou **refactorisation de code** (*refactoring*) :

Pratiquée sans relâche, cette pratique consiste à rendre le code plus « propre » et à améliorer la conception sans en modifier le comportement.

5. Les pratiques collaboratives :

- ➢ **Programmation en binôme** (*pair programming*) : le principe est de faire travailler deux développeurs sur une même machine afin de produire un code de grande qualité, grâce à la relecture « à la source » ; les binômes ne sont pas fixes tout au long du projet, ainsi chacun peut être amené à travailler sur une autre partie de l'application.

- ➢ **Règles de codage** (*coding standard*) : la pratique du pair-programming n'est possible que grâce au respect par tous des règles de codage définies par l'équipe ; cela facilite les interventions croisées.

- ➢ **Propriété collective du code** (*collective code ownership*) : le code n'appartient à personne en particulier ; tous les développeurs sont susceptibles de travailler sur du code dont ils ne sont pas les auteurs originaux.

- **Métaphore** (*metaphor) :* le principe en est de décrire le système et les fonctionnalités par une métaphore, dans le but d'améliorer la communication

- **Intégration continue** (*continuous integration*) : le système est intégralement assemblé et testé une à plusieurs fois par jour, sur une machine indépendante dédiée. Les anomalies sont détectées au plus tôt.

6. Les pratiques relatives à la gestion du projet :

- **Client sur site** (*on-site customer, whole team*) : le client est intégré à l'équipe pour définir précisément les besoins, arbitrer sur les priorités et visualiser « en direct » le résultat des développements. Pour une meilleure communication, le client et les développeurs travaillent dans le même espace autant que possible.

- **élance de planification** (*planning game*) : lors de séances dédiées à la planification de chaque itération, le client définit les fonctionnalités prioritaires. Les développeurs discutent le contenu de ces fonctionnalités, identifient les tâches techniques sous-jacentes, estiment ces tâches et s'engagent sur leur réalisation.

- **Livraisons fréquentes** (*frequent releases*) : la livraison fréquente de versions intermédiaires assure que le produit en cours de développement correspond bien aux attentes. L'intégration continue et les tests réduisent considérablement le coût de livraison.

- **Rythme soutenable** (*sustainable pace*) : l'équipe doit adopter un rythme de travail qui lui permette d'être efficace tout au long du projet ; un développeur fatigué travaille moins bien.

7. Cycle de vie de XP : [6]

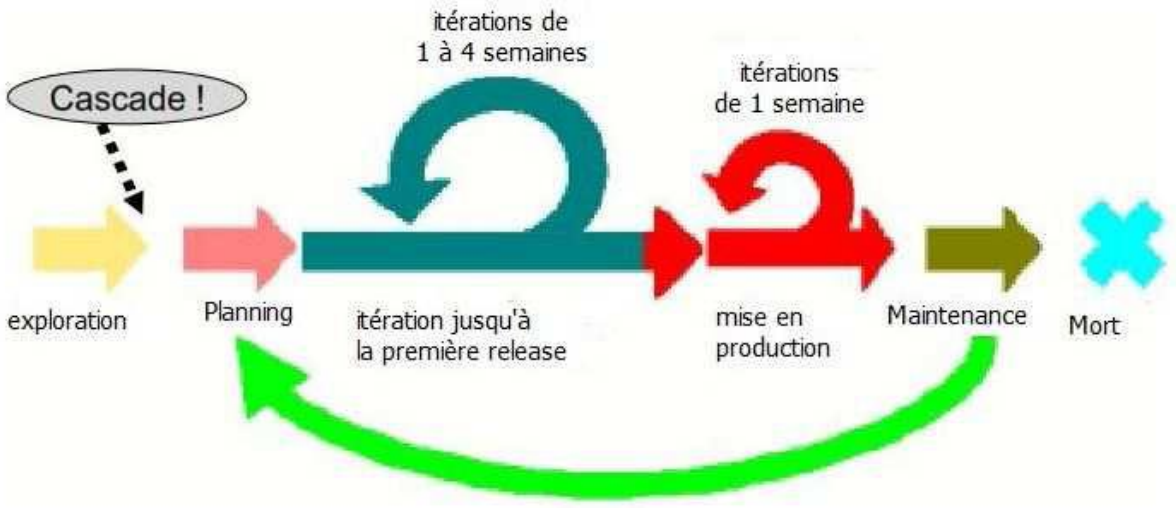


Figure I.2. : Cycle de vie de XP

- **Exploration :**

Exploration des différentes possibilités d'architecture pour le système et étude des limites au niveau des performances présentées par chacune des solutions possibles.
Le client exprime ses besoins en forme de « user stories » que les développeurs devront estimer en termes de temps de développement. Traduit en français, user-stories signifie « histoires de l'utilisateur » et c'est bien ce que ça représente : récit de ce que veut l'utilisateur.
On demande donc au client de définir les spécifications en petites histoires testables en 1 à 2 semaines.
Inévitablement, les besoins ne sont pas suffisamment précis. Le développeur est donc dans l'obligation de dialoguer avec le client pour obtenir les détails ce qui favorise encore une fois l'établissement d'une réelle communication.

- **Planning :**

Le planning de la première livraison est fait de telle sorte qu'un système pourvu uniquement des fonctionnalités essentielles soit mis en production dans un temps minimum et soit enrichi ensuite.
Les séances de planification ou « planning game » durent environ deux jours et la première livraison est souvent programmée pour deux à six mois. Pour ce qui est des procédures entrant dans la phase planning, nous en avons précédemment parlé dans le paragraphe intitulé « séance de planning ».

- **Itérations jusqu'à la première release :**

C'est la phase de développement à proprement parler de la première version de l'application, celle-ci se fait sous forme d'itérations de une à quatre semaines. Chacune de ces itérations relativement courtes produit un ensemble de fonctionnalités passant avec succès les tests fonctionnels associés, ce qui nous permet de revoir le travail aussitôt qu'une déviation par rapport au planning est détectée.

- **Mise en production :**

Pour renforcer le feedback , les itérations de cette phase sont encore plus courtes et des tests parallèles sont conduits au cours de la mise en production et les développeurs procèdent à des réglages affinés pour améliorer les performances (performance tuning).
Le système offre à la fin de cette phase toutes les fonctionnalités indispensables et est parfaitement fonctionnel et peut être mis à disposition des utilisateurs.

- **Maintenance :**

C'est-à-dire continuer à faire fonctionner le système mis en œuvre et lui adjoindre les fonctionnalités secondaires qui avaient volontairement été laissées de coté jusque là, le

développement de ces nouvelles fonctionnalités doit être fait avec prudence de la part des développeurs

- **Mort :**

Lorsque le client n'arrive plus à écrire d'user stories supplémentaires (ce qui signifie que pour lui tous ses besoins ont été satisfaits) ou quand le système est saturé et n'est plus capable de recevoir de modifications pour satisfaire de nouveaux besoins (tout en restant rentable bien sûr) on pourra dire que le projet est fini.

8. Principes de XP : Facteurs *:*

- ✓ Feedback rapide et constant
- ✓ Compréhension partagée
- ✓ Bien-être de l'équipe
- ✓ Processus fluide et continu

VI.2. La méthode Scrum: [7]

1. **Introduction :**

Le Scrum ou « mêlée » est un terme emprunté au rugby.
En rugby, la mêlée a pour objectif « de reprendre le jeu rapidement, en toute sécurité et équitablement, après une faute mineure ou un arrêt de jeu ». Dans la méthode Scrum, le Scrum désigne la réunion quotidienne (Daily Scrum Meeting) qui réunit l'ensemble des acteurs du projet pour dresser, en un temps limité à quinze minutes, le bilan de la journée passée, planifier celle qui commence et repérer les éventuels obstacles rencontrés par chacun. Le Scrum symbolise la solidarité et la force qui lient les membres de l'équipe pour le succès de l'itération.

2. **Historique de Scrum:**

Les racines de Scrum se trouvent dans la publication de « The New Product Development Game ». Cet article, paru en 1986 dans Harvard Business Review, démontre la fin des approches classiques dans le développement de nouveaux produits et met en avant les vertus des petites équipes pluridisciplinaires intégrées et soudées dans une stratégie plus flexible.

3. **Les trois piliers de Scrum :**

Scrum est un processus empirique : il se base sur l'expérience du terrain. Il s'appuie sur trois piliers. Il suit également les principes des méthodes agiles.

- ✓ **La transparence :**

Scrum met l'accent sur le fait d'avoir un langage commun entre l'équipe et le management. Ce langage commun doit permettre à tout observateur d'obtenir rapidement une bonne compréhension du projet.

- ✓ **L'inspection :**

À intervalle régulier, Scrum propose de faire le point sur les différents artéfacts produits, afin de détecter toute variation indésirable. Ces inspections ne doivent pas être faites trop fréquemment, ou par un inspecteur mal formé : cela nuirait à l'avancement du projet.

- ✓ **L'adaptation :**

Si une dérive est constatée pendant l'inspection, le processus doit alors être adapté. Scrum fournit des rituels, durant lesquels cette adaptation est possible. Il s'agit de la réunion de planification de sprint, de la mêlée quotidienne, de la revue de sprint ainsi que de la rétrospective de sprint.

4. Cycle de vie de Scrum :

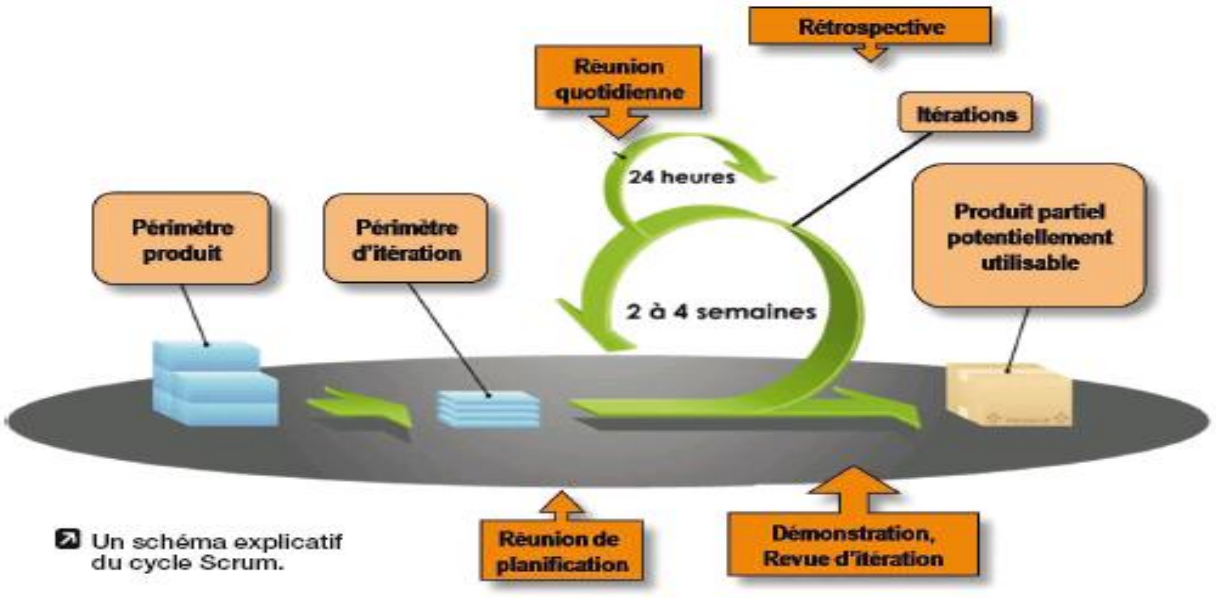


Figure I.3. : Cycle de vie de Scrum.

VI.3. UP: Unified Process – Processus Unifié: [8]

1. Introduction :

Un processus de développement définit une séquence d'étapes, en partie ordonnée, qui concoure à l'obtention d'un système logiciel ou à l'évolution d'un système produire des logiciels de qualité, qui répondent aux besoins des utilisateurs dans des temps et des coûts prévisibles.

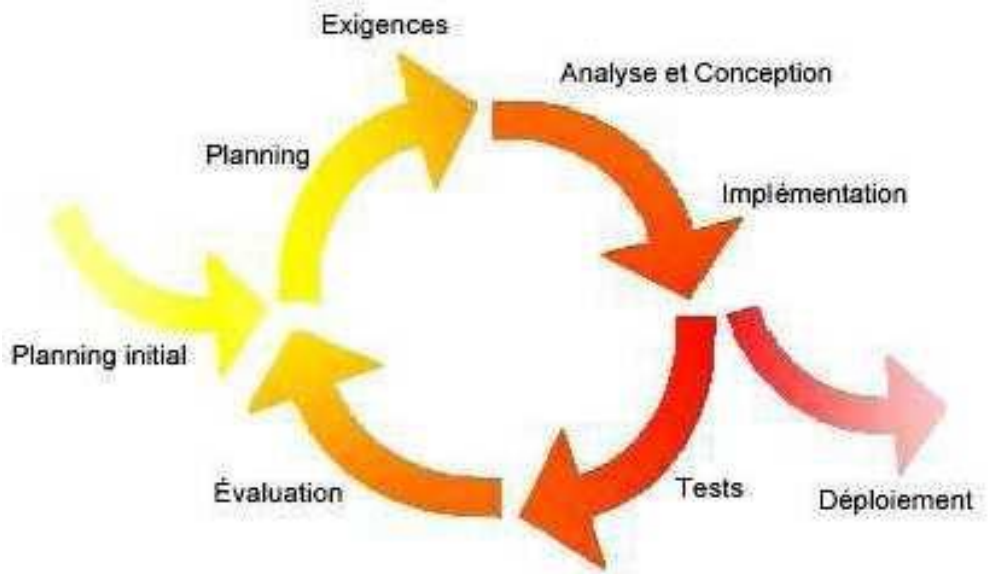
2. Définition :

Processus unifié (PU ou UP en anglais pour **Unified Process**) est une méthode de prise en charge du cycle de vie d'un logiciel et donc du développement, pour les logiciels orientés objets. C'est une méthode générique, itérative et incrémentale, contrairement à la méthode séquentielle Merise (ou SADT).

PU vient compléter la systémique des modèles UML. Elle est le résultat final d'une évolution de l'approche d'Ericsson qui est au fondement d'une des premières méthodes de développement pour applications orientées objets, la méthode Objectory Process (1987).

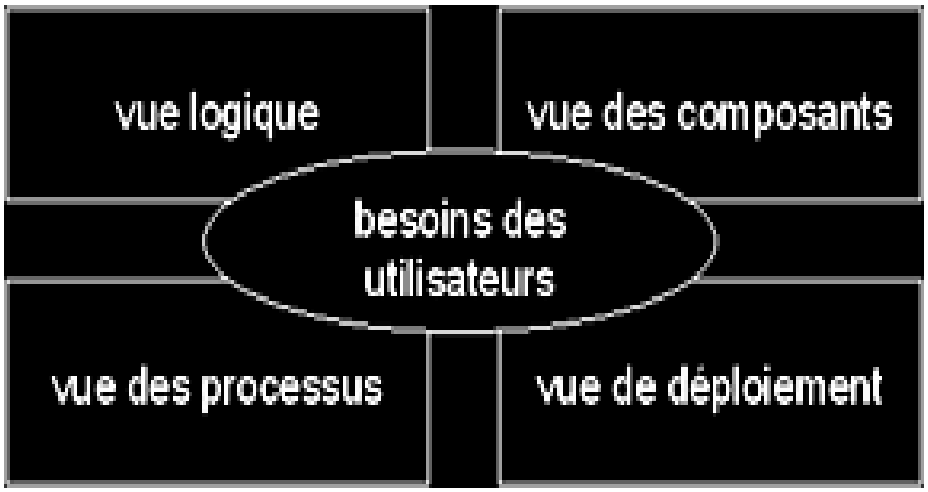
Un processus unifié se distingue par les caractéristiques suivantes :

- **Itératif** : Le logiciel nécessite une compréhension progressive du problème à travers des raffinements successifs et développer une solution effective de façon incrémentale par des itérations multiples.



- **Piloté par les risques** : les causes majeures d'échec d'un projet logiciel doivent être écartées en priorité.

- **Centré sur l'architecture** : le choix de l'architecture logicielle est effectué lors des premières phases de développement du logiciel. La conception des composants des produits est basée sur ce choix.



- **Conduit par les cas d'utilisation** : le processus est orienté par les besoins utilisateurs présentés par des cas d'utilisation.

- **Les activités :**

L'objectif d'un processus unifié est de maîtriser la complexité des projets informatiques en diminuant les risques.
UP est un ensemble de principes génériques adapté en fonctions des spécificités des projets.
UP répond aux préoccupations suivantes :

- **QUI** participe au projet ?
- **QUOI**, qu'est-ce qui est produit durant le projet ?
- **COMMENT** doit-il être réalisé ?
- **QUAND** est réalisé chaque livrable ?

- **Expression des besoins :**

L'expression des besoins comme son nom l'indique, permet de définir les différents besoins :

- inventorier les **besoins principaux** et fournir une liste de leurs fonctions recensées les **besoins fonctionnels** (du point de vue de l'utilisateur) qui conduisent à l'élaboration des modèles de cas d'utilisation

- appréhender les **besoins non fonctionnels** (technique) et livrer une liste des exigences.

Le modèle de cas d'utilisation présente le système du point de vue de l'utilisateur et représente sous forme de cas d'utilisation et d'acteur, les besoins du client.

- **Analyse :**

L'objectif de l'analyse est d'accéder à une compréhension des besoins et des exigences du client. Il s'agit de livrer des spécifications pour permettre de choisir la conception de la solution.
Un modèle d'analyse livre une spécification complète des besoins issus des cas d'utilisation et les structure sous une forme qui facilite la compréhension (scénarios), la préparation (définition de l'architecture), la modification et la maintenance du futur système.
Il s'écrit dans le langage des développeurs et peut être considéré comme une première ébauche du modèle de conception.

- **Conception :**

La conception permet d'acquérir une compréhension approfondie des contraintes liées au langage de programmation, à l'utilisation des composants et au système d'exploitation.
Elle détermine les principales interfaces et les transcrit à l'aide d'une notation commune.
Elle constitue un point de départ à l'implémentation :

- elle décompose le travail d'implémentation en sous-système
- elle créée une abstraction transparente de l'implémentation

- **Implémentation :**

L'implémentation est le résultat de la conception pour implémenter le système sous formes de composants, c'est-à-dire, de code source, de scripts, de binaires, d'exécutables et d'autres éléments du même type.
Les objectifs principaux de l'implémentation sont de planifier les intégrations des composants pour chaque itération, et de produire les classes et les sous-systèmes sous formes de codes sources.

- **Test :**

Les tests permettent de vérifier des résultats de l'implémentation en testant la construction.
Pour mener à bien ces tests, il faut les planifier pour chaque itération, les implémenter en créant des cas de tests, effectuer ces tests et prendre en compte le résultat de chacun.

3. Avantages/Inconvénients :

❖ **Avantages:**

- Itératif
- RUP améliore la qualité du produit.

- RUP améliore la compréhension du système.
- RUP augmente le taux de succès du projet.
- Méthodologie complète
- Intégration avec UML
- Largement adopté en entreprise

❖ **Inconvénients :**

- Coûteux à personnaliser
- RUP ne supporte pas les multi-projets
- RUP exige des experts
- RUP est propriété de Rational Software

VI .4. La méthodologie DSDM (Dynamic Systems Development) : [9]

Dynamic systems Development Method (DSDM) est une méthode de gestion de projet de la catégorie des méthodes agiles. Cette méthode a été développée en Grande-Bretagne à partir de 1994.

DSDM propose une approche globale de gestion de projet, est une des meilleures méthodes « agiles » dans un environnement de développement rapide « RAD ». Les méthodes agiles s'opposent aux méthodes traditionnelles. Comme nous avons vu, pour les approches traditionnelles il faut tout d'abord définir les fonctionnalités des applications, et ces fonctionnalités demeurent figées ou difficilement modifiables. Nous avons aussi vu qu'après la livraison des applications, nous retrouvons toujours les mêmes types de problèmes, il faut encore dépenser des ressources en termes de budget et de temps pour les modifier.

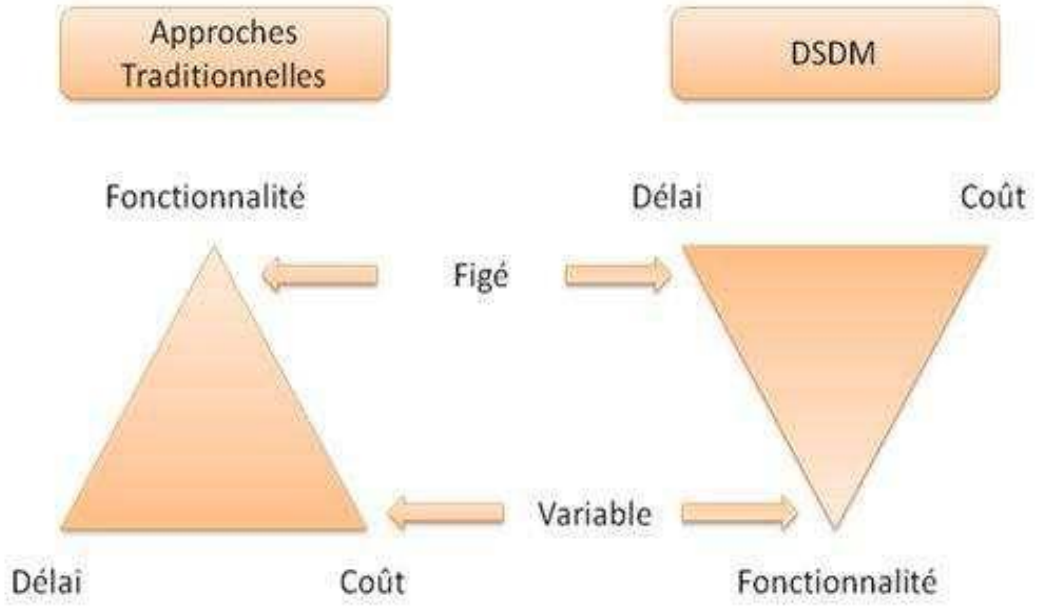


Figure I.4. : La méthodologie DSDM.

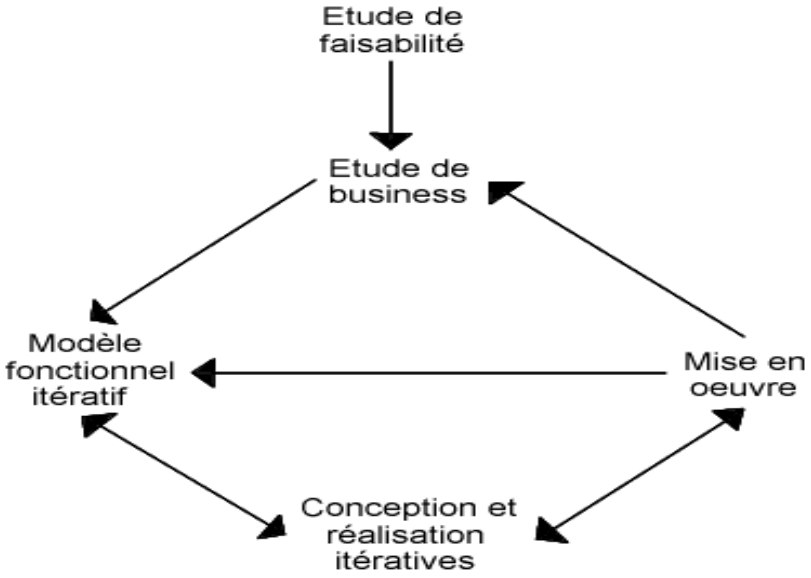
DSDM s'oppose aux méthodes traditionnelles

1. Principes : [9]

La méthode DSDM s'appuie sur 9 principes de base :

- Implication des utilisateurs durant tout le cycle de développement. Ils sont considérés comme des membres à part entière de l'équipe projet.
- Autonomie. L'équipe projet doit avoir un pouvoir de prise de décision concernant l'évolution des besoins.
- Visibilité du résultat. L'application doit être livrée le plus souvent possible afin de permettre un feed-back rapide. Les délais entre les livraisons doivent être le plus court possible.
- Adéquation. L'objectif est de livrer une application en adéquation avec le besoin métier du client.
- Développement itératif et incrémental. L'évolution du développement est basée sur le feed-back des utilisateurs.
- Réversibilité. Toute modification effectuée durant le développement doit être réversible.
- Synthèse. Un schéma directeur défini de manière préalable fixe les grandes lignes du projet, notamment son périmètre.
- Tests. Les tests sont continus durant tout le développement. Ils permettent de garantir le bon fonctionnement de l'application, à chaque étape du développement
- Coopération. Les acteurs du projet doivent faire preuves de souplesse concernant les modifications des fonctionnalités demandées.

- **Processus :**



- Etude de faisabilité : Le but de cette étape est de déterminer s'il est opportun de faire le projet en question. On évalue les coûts, la valeur ajoutée attendue. Dans cette étape, on produit un *Rapport de Faisabilité* ainsi qu'un *Plan Global de Développement.*
  On développe parfois un prototype afin de démontrer la faisabilité technique.

- Etude business (ou analyse fonctionnelle) : Cette étape sert à la définition des spécifications. On définit quelles sont les fonctionnalités que l'application doit apporter, en les priorisant, dans un document appelé *Définition du Domaine Industriel* mais aussi quels types d'utilisateurs sont concernés par l'application, de manière à pouvoir les impliquer. On définit également l'architecture du système, dans un document appelé *Définition de l'Architecture Système* Enfin, à partir du, *Plan Global de Développement*, on définit *un plan global de prototypage*

Modèle fonctionnel itératif : Une fois que les fonctionnalités sont classées par ordre de priorité, on commence à affiner la définition des aspects du système liés au métier.

Les phases Modèle fonctionnel itératif et Conception et réalisation Itératives consistent en un cycle de quatre activités :

1. identifier ce qui doit être fait,
2. décider comment et à quel moment le faire,
3. exécuter,
4. vérifier qu'il a été développé de manière appropriée

- **Conception et réalisation itératives :** C'est une phase où l'on traite les détails et complète le produit sur le prototype fonctionnel. Dans le diagramme du DSDM, le processus ne représente pas la phase de test, mais ceux-ci interviennent durant la phase de Modèle fonctionnel itératif et de Conception et réalisation Itératives. Une fois que

la fonctionnalité d'une partie de projet total est validée par les utilisateurs, on continue à développer les autres, sinon, on recommence la procédure.

- **Mise en œuvre :** Le système est livré avec toute la documentation, y compris la documentation utilisateur et la formation des utilisateurs. C'est une phase qui couvre le passage de l'environnement de développement à l'environnement opérationnel.

2. Avantages, inconvénients et solutions : [9]

Il y a trois types d' « acteurs » pour la méthode DSDM définit : utilisateur, développeur et produit.

Utilisateur :

- **Avantages :**

Pour les utilisateurs, les avantages sont vraiment évidents. Les besoins sont directement exprimés de la part des utilisateurs. Il n'y a pas de non dit entre utilisateurs et les développeurs. Une meilleure confiance s'installe également. Le produit est livré régulièrement : les parties du produit sont progressivement présentées aux utilisateurs, ceux ci connaissent donc déjà le produit ou certaines de ses parties. La mise en œuvre sera plus aisée.

- **Inconvénients *:***

Bien que chaque utilisateur connaisse son métier, il n'est sûr pas qu'il exprime bien ses besoins dans un langage précis que le développeur comprend tout de suite. En plus, chacun a des besoins spécifiques, mais le produit ne peut réaliser toutes les fonctionnalités. Enfin, les utilisateurs doivent gérer leurs tâches quotidiennes et ne sont pas toujours libres de tester le produit et d'en faire le compte rendu.

- **Solutions :**

Le but de DSDM est d'améliorer la qualité du produit et la satisfaction des utilisateurs tout en optimisant le coût et le délai. Il est important que les utilisateurs mettent le projet en priorité et que le support de la direction. Puis, il faut trouver un groupe de gens qui connaissent bien le métier et un peu l'informatique pour faciliter la traduction des besoins. Si ce n'est pas le cas une formation est nécessaire. Quant aux besoins des utilisateurs, la direction de développeur doit les collecter, les classer par ordre de priorité de façon à ce que les plus demandées ou les plus importantes soient développés en premier.

Développeur :

- **Avantage :**

Les besoins viennent donc directement des utilisateurs finals, de plus l'esprit d'équipe et les compétences de coopération du développeur se développent pendant le travail.

- **Inconvénients :**

Le développeur dépend toujours des utilisateurs, il a moins d'indépendance de création ou pour ajouter des fonctionnalités innovantes, il est très sollicité, ce qui réduit sa réactivité, qui est très importante pour ce genre de projet.

- **Solutions :**

Donner aux développeurs plus de droits, et l'encourager à créer des fonctionnalités innovantes, tout faire pour optimiser sa réactivité.

- **Produit :**

- **Avantages :**

Avec le développement itératif en développant, livrant, corrigeant, le produit satisfait progressivement à la plupart des questions des utilisateurs et la mise en œuvre sera rapide.
Evidemment, les utilisateurs acceptent bien mieux le produit qui correspond de mieux en mieux à leurs besoins. Car pendant le développement, une fois qu'il y a un changement, le produit peut être rapidement modifié, le produit est souple face aux changements. De plus la phase de formation est intégrée dans le développement et le groupe d'utilisateurs pilotes seront les formateurs et les promoteurs de l'application.

- **Inconvénients :**

S'il y a beaucoup de changements pendant le développement, le produit se trouverait changé plusieurs fois, cela influencerait le coût et le délai d'une hypothétique recette finale, on peut arriver aussi à des situations complexes, ou aucune des solutions n'est jamais suffisante.

- **Solutions :**

Fixer dés le début ce sur quoi on devra arriver, fixer le coût et le délai, définir toujours la priorité des taches à rendre.

VI.5 Le RAD (Rapid Application Development) : [10]

1. Définition :

La méthode de **développement rapide d'applications**, dite **méthode RAD** (acronyme de l'anglais Rapid Application Development), est la première méthode de développement de logiciels où le cycle de développement est en rupture fondamentale par rapport à celui des méthodes antérieures dites « en cascade ». Ce nouveau cycle qualifié d'itératif, d'incrémental et d'adaptatif, se retrouvera ensuite dans toutes les méthodes dites « agiles » publiées par la suite.
Une méthode RAD devrait, d'après son auteur, apporter trois avantages compétitifs à l'entreprise :

- une rapidité de développement (cette caractéristique donne son nom à la méthode) ;
- un faible coût de développement ;
- une application de grande qualité.

2. Les principes : [10]

Voici les principes de base :

Tout d'abord, le développement devrait être effectué par de petites équipes, expérimentées et ayant reçu toute la formation nécessaire.
Le développement doit s'effectuer en faisant appel à une méthodologie formalisée.
Une équipe de développement RAD doit disposer d'outils puissants qui automatisent le développement dans sa globalité, tant en ce qui concerne les étapes du développement que l'enchaînement de ces étapes.
L'équipe de développement doit être correctement gérée, par un encadrement encourageant la réutilisation des composants, et attentifs aux aspects humains du management de projet.

Enfin, les utilisateurs finaux devraient s'impliquer dans le processus de développement.

La méthode RAD est donc fondée sur quatre ingrédients de base :

- **Les outils :** de conception, de prototypage, de génération de code, disposant d'un langage de haut niveau (L4G), ces outils étant fédérés par un référentiel et constituant un atelier puissant.

- **Les personnes :** elles doivent être compétentes, expérimentées, correctement formées aux techniques et aux outils utilisés, et, cela ira mieux en le disant, ...motivées ! L'auteur revient souvent sur cet aspect fondamental, et un chapitre entier y est consacré. Ce qui est tout aussi essentiel, c'est que l'utilisateur final soit directement impliqué dans chaque phase du projet.

- **Le management :** une gestion du projet, en particulier en ce qui concerne les aspects humains, est essentielle. Le pire ennemi du RAD, c'est la bureaucratie, dit James Martin. Combien de projets n'a-t-on pas vu dériver ou stagner, ensablés par une bureaucratie toujours plus lourde ?

- **La méthodologie :** Rien ne se fera sans une méthodologie, à la fois bien formalisée et souple. L'auteur recommande que la méthode soit consignée, non sur un document qui remplirait la moitié d'une armoire, mais dans un « hyper document » que chacun pourrait parcourir rapidement et de façon non linéaire en fonction de ses besoins et de son rôle sur le projet. Ce document doit être adapté d'un projet à l'autre, toujours affiné, et consultable par tous. Seul un document électronique disponible sur un réseau peut convenir.

3. Les quatre phases :

Le cycle RAD est constitué de cinq phases :

1) **Phase de définition des besoins**. Elle se matérialise par des sessions de travail appelées **JRP** (Joint Requirements Planning ou définition conjointe des besoins).

2) **Phase de « conception utilisateur »** (user design). Elle se caractérise par les sessions **JAD** (Joint Application Development ou développement conjoint d'application), qui est un des signes distinctifs du RAD.

3) **Phase de construction**, mêlant les spécifications détaillées et le codage.

4) **Phase de finalisation** et mise en place (en anglais : cutover).
5) Signalons brièvement deux points essentiels :

4. Avantages et inconvénients de la méthode : [10]

Pour l'utilisateur :

- **Avantages :**

✓ L'utilisateur est placé au premier rang du projet informatique. Il est parti intégrante de l'équipe de projet et participe à tous les stades de l'élaboration du système. Il définit les besoins, les complète et les affine au fur et à mesure de l'avancement du projet. Il décide si des modifications sont à apporter au système.

✓ L'utilisateur est acteur du résultat final.
✓ L'utilisateur connaît déjà le système développé, la formation se fait en parallèle au processus de développement.
✓ L'utilisateur peut travailler directement avec une partie du système. Ce système n'est pas encore terminé, mais est déjà doté des fonctions majeures.
✓ L'utilisateur reçoit dans un intervalle court une application opérationnelle.

- **Inconvénients :**

✓ Le classement des besoins par priorité et le choix entre délais et fonctionnalités n'est pas toujours facile à faire. Cela implique de l'utilisateur un pouvoir de décision et l'acceptation de compromis.

✓ L'implication dans un projet RAD nécessite de l'utilisateur une motivation forte et un investissement personnel supplémentaire.

✓ La livraison d'une version allégée du système peut entraîner frustration et insatisfaction chez l'utilisateur.

- ✓ Une application inachevée risque de le rester faute de nouvelles ressources (hommes, argent, temps).

Pour l'informaticien :

- **Avantages :**

- ✓ L'informaticien-développeur est plus sûr du résultat final, car il l'aura réalisé avec son client.
- ✓ L'implémentation du système est facilitée, car les prototypes sont testés tout au long du cycle de développement.
- ✓ L'informaticien voit plus vite le fruit de son travail.
- ✓ L'informaticien s'enrichit par le contact permanent qu'il a avec son client et apprend à mieux connaître le métier de ce dernier.

- **Inconvénients :**

- ✓ L'informaticien doit faire preuve de discipline lors de la réalisation du prototype. Les besoins prioritaires doivent être traités en premier, autrement les objectifs et les délais ne pourront être atteints.
- ✓ L'informaticien doit passer d'un rôle de décideur à celui de conseiller.

Pour l'entreprise :

- **Avantages :**

- ✓ L'entreprise dispose d'un système de bonne qualité qui respecte les coûts et les délais.
- ✓ L'entreprise dispose d'un système qui répond à ses objectifs au moment nécessaire.
- ✓ Le projet RAD permet à l'entreprise d'accroître les compétences et la formation de son personnel.
- ✓ Le projet RAD rapproche les personnes de métiers différents et favorise une culture d'entreprise.
- ✓ Les risques de projets sans fin et de gouffres financiers sont minimisés.

- **Inconvénients :**

- ✓ La documentation du système peut être insuffisante pour la maintenance et l'évolution future du système.
- ✓ La mise en œuvre de RAD nécessite un investissement important de départ: le personnel doit être formé et disposer de matériels et d'outils performants.

- ✓ L'entreprise doit recourir un consultant indépendant comme animateur/expert RAD.

VI. 6. La méthodologie ASD (Adaptive Software Development) :[11]

1. Définition :

Adaptive Software Development est une méthode agile développée par Jim Highsmith, président d'information Architects, Inc.Elle s'adresse tout particulièrement aux projets e-business. Les plannings traditionnels, associés à un certain degré de certitude et de stabilité du business, ne conviennent plus pour diriger les projets actuels qui doivent être réalisée en des temps très courts, supporter de nombreux changements et incertitudes. Jim Highsmith propose donc avec Adaptive Software Development un cycle de vie plus souple face aux changements qui surviennent entre développeurs, testeurs et client sont deux valeurs essentielles de cette méthode**.**

ASD est une méthode agile de développement logiciel. Elle considère que l'imprévu est inévitable, et que l'on peut en tirer parti pour atteindre de meilleures solutions, qui n'auraient pas été envisagées *a priori*.

Le cycle de vie d'un projet ASD se déroule autour d'une série de cycles en trois volets :

- ❖ **Spéculation :**

- Initier le projet (mission, contraintes, collaborateurs, expression des exigences, identification des risques…).
- Déterminer la durée du projet, le nombre d'itérations et les dates associées (4 à 8 semaines par itération).
- Affecter un objectif (*mission*) à chaque itération.
- Dresser une liste de tâches à réaliser.

- ❖ **Collaboration :**
- Livraison des composants.
- Communication forte et assez informelle.

- ❖ **Apprentissage :**
- Contrôle qualité.
- Suivi et bilan d'avancement.
- Communication forte et assez informelle.

Ses caractéristiques principales sont :

• Focaliser sur l'objectif (*mission focused*) ;
• Se baser sur des composants (*component-based*) ;
• Itérer ;

• Découper le temps et fixer des deadlines (*timeboxing*) ;
• Piloter le projet par les risques (*risk-driven development*) ;
• Accepter le changement.

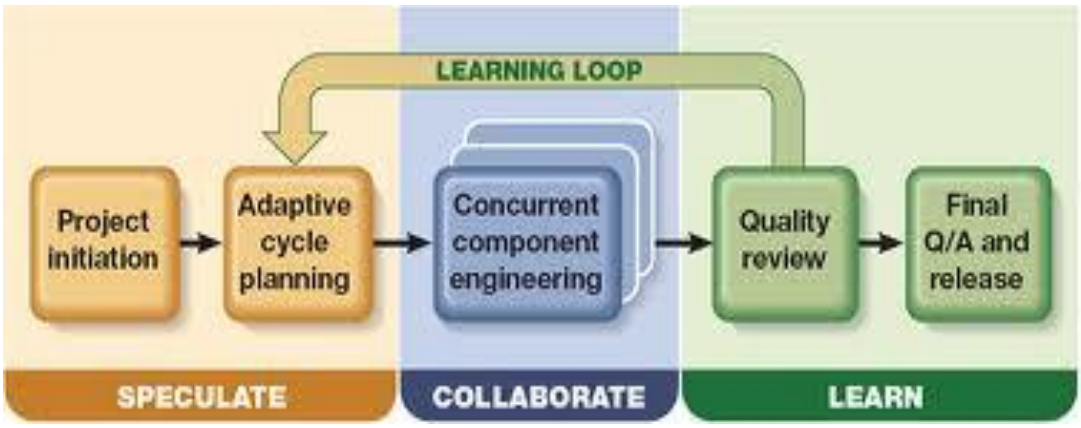


Figure I.5. : Cycle de vie d'ASD.

2. Les 6 caractéristiques principales d'ASD :

Le cycle de vie préconisé par ASD s'appuie sur six valeurs fondamentales :

- Focaliser sur une mission (« mission focused »).
- Se baser sur des composants (« componenet-based »).
- Itérer.
- Découper le temps et fixer deadlines (« timeboxing »).
- Gérer le risque (« risk-driven »).
- Tolérer le changement.

VI. 7. La méthodologie Crystal Clear : [12]

1. Définition :

Crystal clear est une méthode de gestion de projet. Elle est très fortement adaptable aux spécificités de chaque projet

Origine :

* Créée par Alistair Cockburn
* Méthode agile
* La Crystal Family

2. Objectif :

- Lancer rapidement un projet
- Méthode adaptative
- Solidarité et communication d'équipe

3. Les règles de Crystal Clear :

- **Les règles de base**
  - Maximum 6 personnes
  - Travail de groupe
  - Cycle de développement incrémental
  - Un cycle = 3 mois
  - Release fréquente
  - Collaboration avec le client
- **Les normes**
  - Observation des utilisateurs
  - Participation directe du client
  - Des sorties incrémental régulière
  - Automatisation des tests
  - Auto-organisation

4. Environnement :

❖ **Les outils**

- Gestion de version
  - SVN
  - Git
  - ClearCase
- Un open space
- Des tableaux blancs

❖ **L'organisation de l'équipe :**

- Pas de leadership mais un coordinateur
- Une équipe de designer-developers
- Un rôle peut avoir plusieurs sous-rôles
- Un développeur doit : programmer, documenter, tester

5. Cycle de vie :

Le cycle de vie préconisé par Crystal clear s'appuie sur trois processus qui sont :

- ✓ Spécifications
- ✓ Conception et Planning
- ✓ Itérations

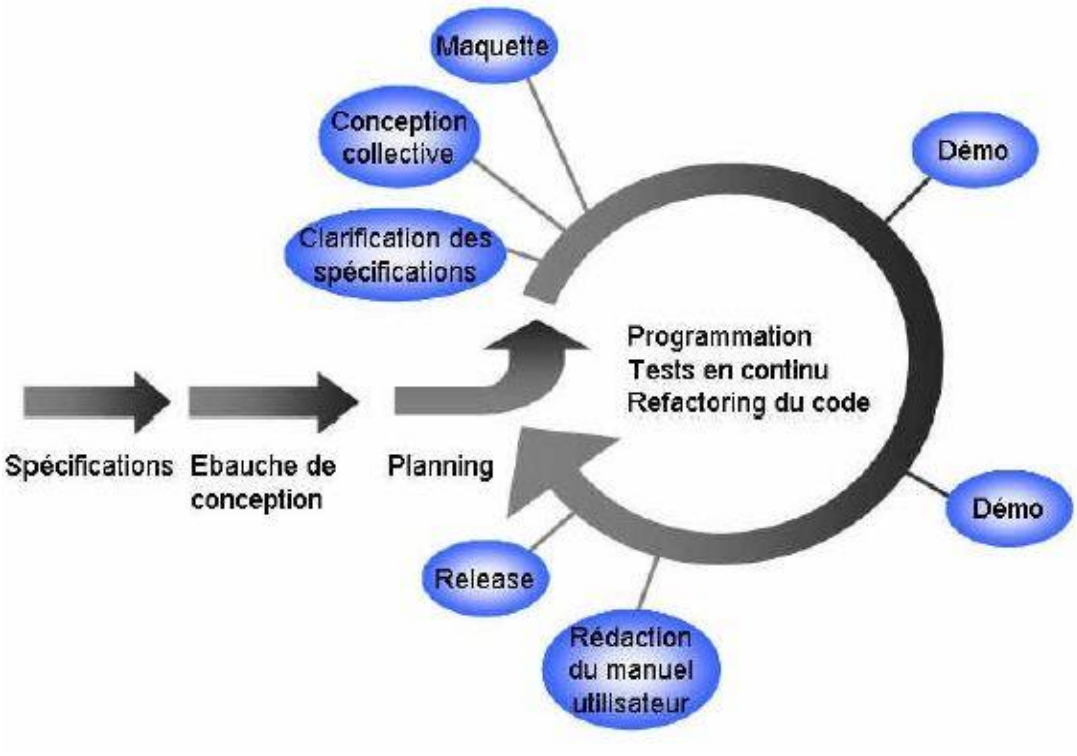


Figure I.6. : Crystal Clear – Heens Romano – 2011-2012.

Spécifications :

- ✓ Observation des utilisateurs.
- ✓ Elaboration des use case.
- ✓ Classification des use case.

Conception et Planning :

- ✓ Choix des technologies.
- ✓ Ebauche.
- ✓ Planification des Release.

Itérations :

- ✓ Travail de conception.
- ✓ Maquettes et démos.
- ✓ Développement libre.
- ✓ Manuel.

6. Les Avantages :

- ➢ Flexibilité.

- Faire simple.
- Livraison fréquente.
- Soucis de la qualité.
- Ecoute permanente du client.

7. Les propriétés de Crystal Clear :

- ✓ Des livraisons fréquentes.
- ✓ Des aménagements permanents.
- ✓ Une communication osmotique.
- ✓ Confiance, liberté d'expression et sécurité personnelle.
- ✓ Focus sur l'objectif et disponibilité.
- ✓ Un contact permanent avec les utilisateurs.
- ✓ Un environnement de travail approprié pour l'automatisation des tests, la gestion de configuration et les intégrations fréquentes.
- ✓ Une collaboration étroite entre toutes les parties prenantes, y compris en dehors de l'équipe.
- ✓ Une réflexion constante sur ces propriétés.

VI.8. La méthode FDD (Feature Driven Development) : [13]

1. Définition :

Feature Driven Development est une méthode agile développée par Jeff de Luca et Peter Coad pour mieux gérer les risques existants dans les projets de développement. L'idée de base de la méthode est de procéder par itérations très courtes, de deux semaines, avec production à la fin de chaque itération d'un livrable fonctionnel. Les spécifications sont découpées le plus possibles en caractéristiques simples : les « features » que l'on peut regrouper en « features sets ».

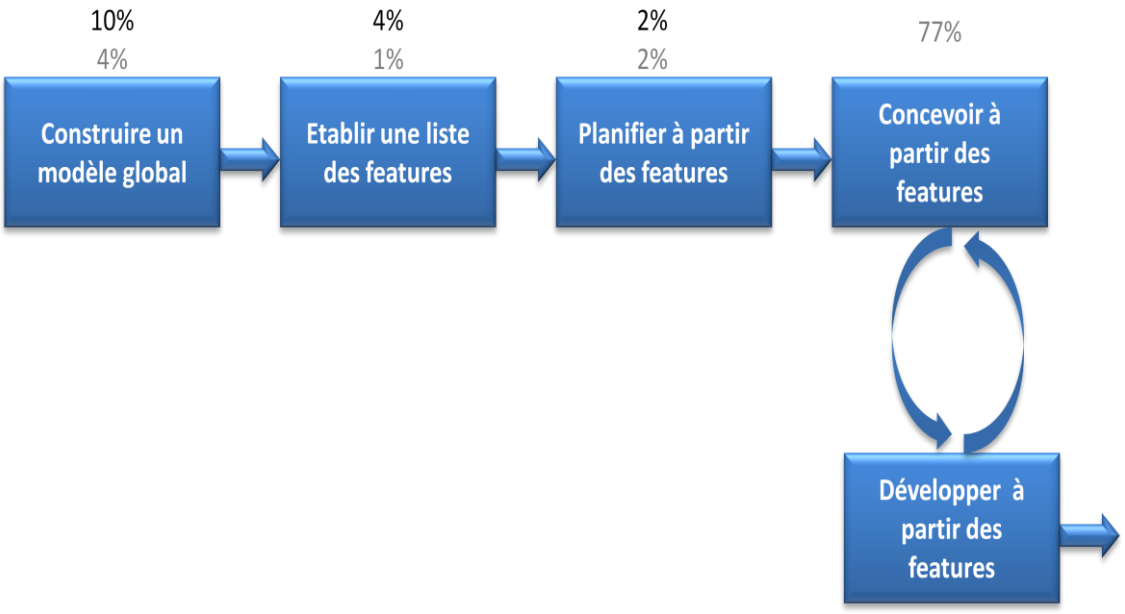
Pour les développeurs, procéder par itérations est très motivant puisqu'ils doivent être capables de livrer toutes les deux semaines des composants qui sont utiles au client, qui fonctionne et sont porteurs de valeur pour lui.

Du point de vue des managers, cette façon de procéder permet de connaitre facilement l'état d'avancement du projet et d'avoir du concret à montrer au client à chaque fin d'itération ; Le fait de délivrer fréquemment des composants fonctionnels permet aussi une bonne gestion du risque. Les clients ont une vue claire sur le planning et les différentes étapes du projet, ils ont aussi des résultats concrets qui leur prouvent l'avance du projet.

2. Les 5 phases d'un projet FDD :

- **Phase 1 :** Développer le modèle global.
- **Phase 2 :** Etablir une liste détaillée de features classées par priorité.
- **Phase 3 :** Planifier à partir des features**.**

- **Phase 4 :** Concevoir à partir des features.
- **Phase 5 :** Construire à partir des features **.**



Chiffre du haut : pourcentage de la partie linéaire (phase1 à 3)

Chiffre du bas : pourcentage de la partie itérative (phase 4 à 5)

Linéaire Itératif

3. Forces et Faiblesses :

FDD présente les avantages des méthodes agiles, à savoir gestion des risques, flexibilité par rapport au changement, rapidité, livraisons fréquentes etc.

FDD a déjà été utilisé avec des équipes de taille conséquente pouvant aller jusqu'à une vingtaine de développeurs. Le facteur limitant cependant la taille d'une équipe est le nombre de développeurs seniors à disposition.

La propriété du code revenant aux propriétaires de classes a l'avantage de ne permettre des modifications que par une personne qui a une vision globale d'une classe. On peut opposer à cela le principe de propriété collective du code en vigueur dans eXtreme Programming qui nécessite une plus grande discipline mais peut s'avérer plus rapide si un développeur a besoin de modifier du code écrit par un autre membre de l'équipe.

L'inspection du code permet d'apporter un regard neuf sur chaque portion du code et permet ainsi de produire un code de meilleur qualité. Cependant, la personne qui inspecte le code ne le connait pas a priori et il lui faut donc un minimum de temps pour se familiariser avec ce code. Encore une fois, il peut s'avérer plus judicieux de passer à la programmation en binôme pour éviter cette perte de temps et faire la révision du code en même temps que son écriture.FDD n'est d'ailleurs par fermée à ce genre de pratiques qui poussent l'agilité encore plus loin.

VII .Avantage et inconvénients des méthodes agiles : [14]

- **Avantages :**

Apport de valeur ajoutée :

Les exigences sont la propriété du client ou de son représentant, qui les valorise et les hiérarchise, en fonction de la valeur ajoutée que leur implémentation apporte à l'organisation. La planification et le pilotage du projet sont basés sur cette hiérarchisation, susceptible d'être modifiée au cours du projet ; ce qui, finalement, amène l'équipe à livrer, en continu, de la valeur ajoutée à son client.

Adaptabilité :

Grâce au développement itératif et au recueil permanent du feedback du client, l'équipe agile est en mesure d'aligner continuellement le produit développé sur les besoins exprimés et précisés par le client au fil du projet. Cette capacité à s'adapter à s'adapter à l'évolution des exigences est la démonstration de son agilité.

Visibilité :

En mesurant et en évaluant l'avancement du projet sur le nombre de fonctionnalités réellement implémentées et validées par le client, et en analysant en permanence l'adéquation du processus, la visibilité est accrue, tant sur le travail effectué que sur le travail restant à faire .La planification et les arbitrages nécessaires sont donc facilités, dans un contexte consensuel , de surcroît.

Réduction des risques :

Motivée par la livraison de valeur ajoutée pour le client, soucieuse de démontrer son adaptabilité et guidée par une meilleure visibilité, une équipe agile réduit les risques d'échec du projet. Grâce au feedback permanent, les dérives ou les dysfonctionnements sont détectés précocement et peuvent être amoindris, par l'acceptation du changement.

- **Inconvénients :**

Cette approche permet de réduire l'incertitude en réduisant le délai de livraison. Mais, c'est bien beau de le dire dans une phrase. Mais, il aura aussi des impacts sur l'organisation tant du travail.
Dans les approches traditionnelles, la livraison s'effectuait en fin de projet. Mais, avec l'approche la plus part des approches Agiles, l'équipe de projets livrent une partie du logiciel dans un délai court, 2-4 semaines et surtout à rythme soutenu. Donc, il faut aussi en prendre conscience.

VIII. Conclusion :

Dans ce chapitre nous avons présenté une vue générale les différents méthodes agiles qui permettent de bien contrôler la qualité de chaque partie et surtout améliore l'acceptation de chaque partie des utilisateurs. Le développement du projet est aussi le développement de la connaissance des utilisateurs; Dès que le projet est terminé, les utilisateurs, connaissant déjà le produit, peuvent facilement l'utiliser.

L'objet du suivant chapitre sera sur les généralités des assurances.

CHAPITRE II :

Généralités sur les assurances

I. Les assurances

I.1. Introduction :

Ce chapitre porte sur les assurances en général, Il couvre plus spécifiquement les sociétés d'assurance en Algérie et à l'étranger leurs principes et leurs méthodes, leurs avantages et inconvénients et d'autres titres en particulier Groupe Axa.

I.2. Historique : [15]

Contrairement à ce que pourrait penser la majorité des gens, l'histoire de l'assurance est relativement récente, elle repose sur une technique mathématique dont les bases n'ont été élaborées qu'au XVIIème siècle.
Au début, c'est l'assurance maritime qui va se dégager ; le commerçant garantissait la perte de la valeur du navire et de sa cargaison en cas de sinistre, contre le paiement préalable d'une certaine somme.

Vient ensuite le tour des assurances terrestres qui apparaissent au XVIIIème siècle comme un phénomène de civilisation liée aux nouvelles conditions de vie des pays urbanisés et industrialisés. La naissance de l'assurance terrestre est liée au phénomène urbain ; le célèbre incendie de Londres de 1666 qui détruisit 13.000 maisons et 100 églises dans un quartier de 400 rues a suscité la création des premières compagnies d'assurance contre l'incendie.

Les assurances sur la vie apparaissent au Moyen Age avec l'apparition des premières mutualités dans le cadre des corporations d'ouvriers. Plus tard, à la fin du XVIIème siècle et au XVIIIème siècle, c'est un banquier napolitain qui, en 1653, avait conçu la création de groupements d'adhérents constitués sur une durée déterminée, fixée à 15 ans le plus souvent. La capitalisation des cotisations des adhérents ouvre la voie de l'assurance sur la vie.

L'évolution de l'assurance au XIXème siècle est étroitement liée aux modifications des conditions de vie suite à un essor démographique et un exode rural d'une part, et à un déclin de la solidarité familiale d'autre part. Les sociétés d'assurance sur la vie et les mutuelles se multiplient. Par ailleurs, l'industrialisation et la mécanisation provoquent de plus en plus d'accidents.

A partir du XX ème siècle, de nouveaux concepts sont apparus, on parle alors de l'obligation de l'assurance, en matière de responsabilité civile, et de sécurité sociale qui représente une véritable nationalisation de certaines branches de l'assurance, en matière d'accidents de travail, maladie, maternité, invalidité, décès et retraite.

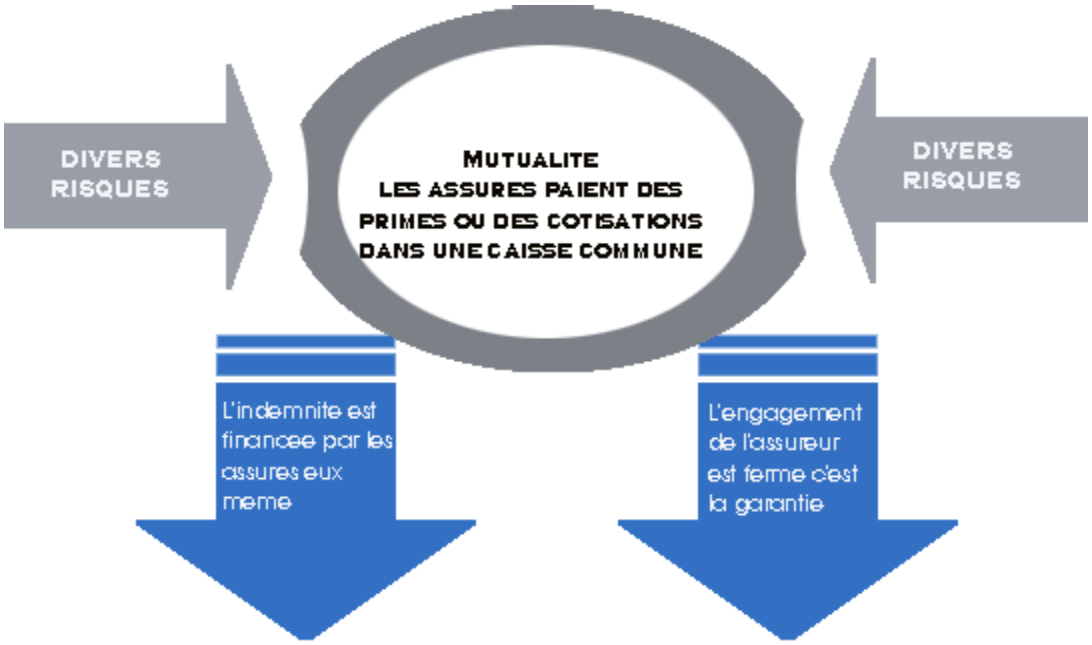
I.3 Définition : [15]

L´assurance est l´activité qui consiste, en échange de la perception d´une cotisation ou prime, à fournir une prestation prédéfinie, généralement financière, à un individu, une association ou une entreprise lors de la survenance d'un risque.

Cette assurance est souscrite auprès d´une société qui a peut en faire son activité exclusive (compagnies d´assurances) mais qui peut aussi en faire une activité complémentaire (banques,...).

La définition complète de l'assurance sera donc la suivante :

L'assurance est opération par la quelle une personne (l'assureur) groupe en mutualité d'autres personnes (les assurés) afin de les mettre en situation de s'indemniser mutuellement des pertes éventuelles (les sinistres) auxquelles les expose de la réalisation de certains risques, aux moyen de sommes (primes ou cotisations) versés par chaque assuré à une masse commune gérée par l'assureur.



I.4. Les différents types d'assurances : [16]

Il existe deux grandes catégories d'assurances : celles qui couvrent une personne physique et celles qui couvrent les biens. Mais, il est également possible de souscrire plusieurs assurances dans un même contrat. On parle alors de « multirisques ».

- **L'assurance de personnes :**

Une assurance de personnes a pour objet de couvrir les risques relatifs aux individus comme les accidents corporels, la maladie, le décès ou encore l'invalidité.

On distingue la prévoyance (garantie emprunteur, indemnités journalières, rente éducation…) et la santé laquelle est subdivisée en deux catégories bien distinctes : la garantie obligatoire (Sécurité sociale) et la garantie complémentaire (mutuelle, assureurs…).

L'assurance de personnes peut être souscrite soit à titre individuel soit à titre collectif. Certains contrats permettent la constitution et le versement d'une épargne sous forme de capital ou de rente. C'est notamment le cas d'une assurance vie.

- **L'assurance des dommages :**

L'assurance des dommages permet d'obtenir une indemnisation en cas de sinistre**.**

Elle regroupe à la fois la protection de responsabilité (responsabilité civile, responsabilité civile familial) le ou responsabilité professionnelle) et celle de biens (dommages causés au véhicule, protection des biens meubles ou immeubles).

Par exemple, en cas d'accident de la route, elle garantit entre autres l'indemnisation des dommages subis par la voiture et s'avère donc nécessaire même si, dans la plupart des cas, elle n'est pas obligatoire. C'est notamment le cas de la prévoyance.

On distingue deux niveaux de garanties dommages : la garantie dommages collisions (permettant à un assuré de bénéficier d'une indemnisation en cas d'accident responsable avec la présence d'un tiers identifiable) et la garantie dommages tous accidents (permettant à un assuré de bénéficier d'une indemnisation en cas d'accident responsable même en l'absence de tiers).

I.4.1.l'assurance vie : [17]

1. Principe :

Il existe plusieurs types d'assurance vie : l'assurance vie temporaire, l'assurance vie entière et l'assurance vie universelle. Voici le fonctionnement de chacune d'elles.

- **L'assurance vie temporaire :**

Cette assurance procure une protection pour une période limitée, par exemple 5, 10 ou 20 ans. On appelle respectivement ces assurances T5, T10 et T20, le « T » signifiant « temporaire ». L'assurance vie temporaire est habituellement renouvelable à des prix qui augmentent de période en période. Par exemple, une T10 offrant un montant d'assurance de 100 000 $ pourrait vous coûter annuellement 126 $ jusqu'à l'âge de 30 ans, 163 $ de 31 à 40 ans, 268 $ de 41 à 50 ans, 626 $ de 51 à 60 ans, etc. Dans les premières années du contrat, pour des assurés jeunes, cette assurance coûte beaucoup moins cher que l'assurance vie entière. Toutefois, si l'assuré vit longtemps, le coût de l'assurance augmente beaucoup. Dans ce type d'assurance, vous payez pour vous assurer et rien de plus. Il n'y a aucune épargne rattachée à cette forme d'assurance. Ainsi, si vous vous assurez pendant 15 ans et choisissez de ne pas renouveler le contrat, vous aurez peut-être l'impression d'avoir payé pour rien, car l'assureur ne vous remettra aucune somme d'argent. L'assurance vie temporaire sert pour les besoins d'assurance temporaires, par exemple pour qu'en cas de décès, le montant de l'assurance puisse :

- ✓ Permettre de rembourser le solde d'un prêt;
- ✓ Aider financièrement les enfants tant qu'ils sont dépendants des parents.

Il existe aussi sur le marché des assurances temporaires, pour 100 ans (T100). Elle est souvent considérée comme une assurance couvrant les besoins d'assurance pour la vie entière, puisqu'elle offre une couverture jusqu'à l'âge de 100 ans.

- **L'assurance vie entière :**

Cette assurance s'applique jusqu'au décès de l'assuré et sert notamment aux gens qui désirent laisser un héritage. En la conservant, vous êtes certain qu'un jour ou l'autre, l'assureur paiera le montant d'assurance prévu. Les primes sont généralement fixes. Certaines assurances prévoient que les primes ne seront payables que pendant un certain nombre d'années, par exemple 10 ou 15 ans. Dans ce cas, vérifiez si cela est garanti. Si vous conservez votre assurance jusqu'à votre décès, le montant qui sera versé par l'assureur est toujours libre d'impôt. Ainsi, payer des primes d'assurance pour ce type de produit, c'est comme si vous investissiez en sachant quel sera le montant versé à votre décès. Ce qu'on ne sait pas c'est le nombre d'années pendant lesquelles vous paierez les primes et quand le montant prévu sera versé au bénéficiaire.

Les assurances couvrant des besoins permanents comportent généralement des valeurs de rachat. Une valeur de rachat est le montant que vous obtenez en mettant fin à votre assurance. Attention ! Avant de mettre fin à une assurance pour obtenir la valeur de rachat, informez-vous auprès de votre représentant des conséquences financières et fiscales. Il est parfois possible d'emprunter sur cette valeur de rachat et ce, sans mettre fin à l'assurance. Attention ! Il s'agit alors d'un prêt : vous devrez le rembourser avec intérêts faute de quoi, votre assurance prendra éventuellement fin.

❖ **L'assurance vie universelle :**

Contrairement à son nom, cette assurance ne s'adresse pas à tout le monde. Elle combine à la fois une portion assurance et une portion placement. Ainsi, vous pouvez payer des primes plus importantes que le coût d'assurance. Par exemple, le coût d'assurance est de 400 $ par année, mais l'assureur vous permet de débourser jusqu'à 1 200 $. Le surplus que vous payez s'accumule dans votre fonds de capitalisation à l'abri de l'impôt tant que vous ne retirez pas les sommes. Vous pouvez choisir d'investir les sommes parmi plusieurs types de placements, par exemple dans des fonds distincts. Le montant accumulé peut servir à payer les coûts d'assurance. Vous pouvez également retirer des sommes si vous le désirez. De plus, vous n'êtes pas tenu de payer une prime d'assurance chaque année. Vous devez toutefois vous assurer qu'il y a suffisamment d'argent dans le fonds de capitalisation pour payer le coût d'assurance.

L'assurance vie universelle se vend sous deux formes : avec des primes qui augmentent avec l'âge ou des primes fixes.

2 .Avantages :

Le principal avantage de l'assurance vie est d'être un véhicule, à la fois d'assurance et de retraite. Elle peut être également un véhicule de succession et de donation.

- **Un contrat d'assurance vie** : les contrats d'assurance vie, ou police d'assurance, garantissent à leur titulaire le versement à la personne désignée sur le contrat, le bénéficiaire, des sommes versées au contrat. Le titulaire peut désigner qui bon lui semble en tant que bénéficiaire. Il peut également désigner une ou plusieurs personnes et leur attribuer des parts sur le capital. A défaut d'une désignation expresse, nominative du ou des bénéficiaires, le contrat stipulera une clause de style attribuant les capitaux versés sur le contrat au conjoint, à défaut aux enfants, et à défaut aux héritiers. Cela signifie qu'au cas où le conjoint serait déjà décédé au jour de la mort du bénéficiaire, les enfants deviendraient les bénéficiaires. S'il n'y a pas ou plus d'enfants en vie, les fonds seront versés aux héritiers du titulaire. L'assurance vie est donc un instrument de succession.

- **Un véhicule de succession** : l'intérêt d'utiliser l'assurance vie pour préparer sa succession est principalement fiscale, puisque le versement des fonds au décès du titulaire est hors des droits de succession dans la limite d'un capital de 152 000 euros. Au-delà de ce plafond, l'article 990-I du Code général des impôts soumet ce contrat à une taxation forfaire de 20 %. En outre, l'article 757-B du CGI impose les primes versées par un titulaire âgé de plus de 70 ans effectués depuis le 20 novembre 1991, si celles-ci dépassent la somme de 30 500 euros.

- **Un mécanisme de donation** : Habituellement, le titulaire se désigne comme bénéficiaire du contrat en cas de vie, c'est-à-dire comme bénéficiaire au jour de la retraite. La chose est logique et fait de l'assurance un contrat d'épargne retraite. Mais le bénéficiaire peut également désigner un tiers comme bénéficiaire, même en cas de vie au jour de la retraite. De la même manière que pour la succession, il s'agit alors d'une donation hors droit de donation, non taxée en deçà des 152 000 euros de l'article 990-I.

- **Une fiscalité attrayante** : même si l'assurance vie présente moins d'avantages que par le passé, elle reste une enveloppe fiscale de choix. D'une part, les produits et les intérêts capitalisés au contrat sont hors imposition durant toute la vie du contrat. D'autre part, au bout de huit ans, les rachats et retraits effectués sur le contrat sont exonérés d'impôt. Cependant, les retraits effectués pendant les quatre premières années sont taxés à hauteur de 35 % et de la cinquième à la huitième année à 15 %. Quoi qu'il en soit, tout retrait, même après huit ans, reste soumis aux prélèvements sociaux de 11 %.Les plus values sont quant à elles taxées à hauteur de 7,5 % à la sortie du contrat, contre les 27 % du droit commun. En outre, cette imposition ne concerne que les intérêts capitalisés à compter du 1 janvier 1998, sur tous les contrats pour les versements effectués à partir du 26 septembre 1997. Cette fiscalité s'applique après un abattement annuel de 4 600 euros pour une personne seule et de 9 200 euros pour un couple marié. Donc en jouant astucieusement de ce plafond annuel de 4 600 euros de retrait, il est possible d'échapper au 7,5 % d'imposition pour tout retrait ou rente pendant votre retraite.

3. Inconvénients:

- **Les frais de gestion**, les frais d'entrée et d'arbitrage sont généralement assez élevés. Les frais de gestion sont variables selon les contrats. Les frais de gestion les plus bas [2]aujourd'hui sont de 0,36 % pour les fonds en euros. Les frais d'arbitrage sont au minimum de 1%, et les frais de versement vont jusqu'à 5% dans les grandes compagnies d'assurance.

- **En cas de divorce,** le contrat d'assurance vie, quel que soit le titulaire, ne fait pas obstacle à la liquidation de la communauté des époux s'il y en a une. En ce cas, il sera nécessaire de racheter le contrat pour partager les fonds. Or, si le divorce et la liquidation interviennent avant la période de 8 ans, le capital sera amputé de la taxation.

4. Contrat d'assurance-vie : [17]

L'assurance-vie est un contrat passé entre une compagnie d'assurance (ou une banque) et vous. En vertu de ce contrat, vous vous engagez à verser des sommes appelées primes à l'assureur. En contrepartie, ce-dernier versera au bénéficiaire du contrat un capital ou une rente en cas de décès ou à une date prévue dans le contrat. L'assurance-vie est utilisée comme un moyen de placement car elle bénéficie d'une fiscalité avantageuse.

5. Types de contrat d'assurance-vie : [18]

En règle générale, le contrat d'assurance-vie permet de constituer une réserve d'argent utilisable à long terme. Les modalités de reversement du capital ainsi constitué varient selon la forme choisie pour le contrat.

- **Les contrats en cas de décès** : le capital ou la rente est versé à un bénéficiaire désigné dans le contrat lorsque l'assuré décède. Ces sommes sont perçues par les bénéficiaires hors succession, ce qui signifie qu'elles n'entrent pas dans le calcul de l'héritage.
- **Les contrats en cas de vie** : ils permettent aux assurés d'épargner en se constituant un capital qui leur sera reversé sous forme de capital ou de rente. Ce système est très avantageux fiscalement mais si l'assuré décède avant la date convenue, l'assureur ne lui devra rien.
- **Les contrats vie et décès** : ce sont les plus fréquents. A la date prévue dans le contrat, les fonds sont reversés à l'assuré s'il est toujours vivant, à son bénéficiaire s'il est décédé.

6. Les 3 éléments du contrat d'assurance :

Le contrat d'assurance comprend trois éléments: le risque, la prime et la prestation.

- **Le risque :**

C'est un élément fondamental de l'assurance, et nous pouvons le définir comme étant un incertain et qui ne dépend pas de la volonté des parties et spécialement de celle de l'assuré.

- **La prime :**

IL s'agit de la prime d'assurance, montant versé par l'assuré à l'assureur en contrepartie du risque pris en charge.

- **La prestation de l'assureur :**

Il s'agit de l'indemnisation de l'assuré, montant touché par l'assuré cas de la survenance du risque pris en charge.

La particularité du contrat d'assurance, c'est le phénomène aléatoire qui caractérise la réalisation du risque (sinistre), donc le paiement de la prime est effectif alors que le versement de la prestation reste aléatoire, c'est en ce sens que le contrat d'assurance c'est un contrat aléatoire.

La seule condition à l'assurabilité d'une chose est le risque (ou aléa), c'est-à-dire l'imprévisibilité d'un événement dommageable. En conséquence, sur le principe, il est possible de souscrire une assurance pour tout événement relatif à la propriété d'un bien meuble, à celle d'un bien immeuble, à la vie, à la santé, etc...

1.5. Rôle de l'assurance :[19]

L'assurance joue des rôles multiples:

1- Premièrement, l'assurance remplit les fonctions de sécurité étant du point de vue individuel que du point de vue général:

- **Au regard de l'assuré :** l'assurance a un caractère moral. En effet, elle est le produit de la vertu de prévoyance. Au lieu d'attendre d'être frappé par les coups du sort et de se trouver ensuite plus ou moins à la charge de la société, l'assuré prend des précautions: il songe à l'avenir ; et à l'avance, de façon constante, il fait, volontairement, un sacrifice personnel pour se prémunir contre le hasard. Il y a même certaines assurances où l'assuré agit, non pour lui-même, mais dans l'intérêt d'autrui, de façon désintéressée ou tout au moins pour accomplir un devoir moral.

En dehors de cette vertu morale, l'assurance a pour rôle fondamental de conférer aux assurés la sécurité dont ils ont besoin. Elle leur apporte la confiance dans l'avenir: grâce à elle, ils sont protégés contre les risques du hasard, qui les menace, eux ou leur patrimoine. L'assurance répond à un besoin incontestable de l'individu: exposé aux coups du sort dans sa personne ou dans ses biens, il ne peut agir d'une façon pleine et efficace que s'il peut se prémunir contre L'aléa qu'il redoute.

Ce besoin de sécurité individuelle auquel répond l'assurance est d'autant plus grand aujourd'hui que la vie moderne se caractérise par un accroissement des risques, donc par une augmentation de l'insécurité. L'assurance devient ainsi, de nos jours, une véritable nécessité pour l'homme, spécialement pour l'homme d'action ou l'homme d'affaire exposé professionnellement à de multiples risques (incendie, vol, responsabilité) contre lesquels il est obligé de se protéger.

- L'assurance, sur le plan de la sécurité, présente un intérêt général et social. En donnant la sécurité aux individus, l'assurance renforce l'économie nationale: elle devient un facteur de production. Elle permet, en effet, de conserver les forces productives, travail et capital, tout au moins de les reconstituer aisément et, à cet égard, elle accroît d'autant plus la puissance économique que les biens nouveaux substitués aux biens détruits peuvent être d'un rendement supérieur. Elle accroît aussi cette puissance par son action préventive en incitant ou même en obligeant les assurés à utiliser les procédés les plus perfectionnés.

2- Deuxièmement, l'assurance permet, par l'accumulation des primes, la constitution des capitaux :

En effet, grâce à l'assurance, des sommes généralement modiques et qui, sans cela, auraient été vraisemblablement consommées, sont réunies au sein de l'entreprise, conservées, placées jusqu'au jour ou elles doivent servir au règlement des sinistres.

3- Troisièmement, l'assurance remplit une fonction de crédit tant au profit des assurés que de l'économie générale:

- elle est tout d'abord, et sous des formes diverses, un moyen de crédit pour l'assuré. Elle facilite en premier lieu son crédit en renforçant les garanties qu'il offre à ses créanciers. Ainsi un débiteur hypothécaire est pratiquement obligé, par une clause de style, d'assurer contre l'incendie, l'immeuble hypothéqué, afin de donner à son créancier la certitude d'être indemnisé au cas où l'immeuble serait détruit par le feu.
- L'assurance joue un rôle non négligeable au regard du crédit général. Les compagnies d'assurance sont obligées de constituer des réserves (provisions) et de les représenter en partie par les titres émis par l'Etat et les collectivités publiques, de sorte que, par leurs placements imposés, elles soutiennent le crédit général du pays.

4- Quatrièmement, l'assurance joue un rôle international.

Ce rôle se réalise de deux façons. D'une part, il appartient aux compagnies nationales de souscrire directement des assurances à l'étranger; d'autre part et surtout c'est par la réassurance que se réalise le rôle international de l'assurance: après avoir traité directement avec ses assurés, l'assureur rétrocède, soit facultativement, soit obligatoirement (traité de réassurance), une partie de ses risques à un réassureur, le plus souvent étranger, de sorte que les incidences des sinistres nationaux se répercutent en définitive sur l'économie de plusieurs pays, ce qui est un facteur d'équilibre et de stabilité générale.

II. Les compagnies d'assurances : [20]

II.1 Définition :

La compagnie d'assurance participe à l'ensemble des activités économiques de notre société. Elle est :

- ✓ Marchand de sécurité.
- ✓ Financier.
- ✓ Prestataire de services.

C'est donc un instrument de sécurité, de crédit et d'épargne. Il ne faut enfin pas oublier que les sociétés d'assurance, par le biais des taxes perçues sont collecteurs d'impôts et de plus un employeur très important du secteur tertiaire.

II.2 Les principaux risques auxquels s'expose une compagnie d'assurance :

L'activité d'assurance est par nature une activité risquée : la fonction économique globale des entreprises d'assurance est d'assumer des risques qu'une personne physique ou morale ne pourrait supporter en mutualisant un grand nombre de risques similaires. Néanmoins, une entreprise d'assurance est elle-même exposée à un certain nombre de risques globaux qui peuvent menacer son existence et, dans des cas extrêmes, la ruiner. Pour cela, la société d'assurance doit être en mesure de se prémunir contre les risques suivants :

- ✓ Les risques techniques ;
- ✓ Les risques de comportement (des assurés et des distributeurs) qui incluent notamment le risque de liquidité ;
- ✓ Les risques de placement ;
- ✓ Les risques de solvabilité (comptable).

Ces risques sont fortement dépendants. Pour se couvrir contre leur réalisation, il est nécessaire d'avoir une vision d'ensemble et de ne pas traiter séparément le passif et l'actif.

II.3 Les sociétés d'assurances en Algérie :

II.3.1. Evolution de l'assurance en Algérie : [21]

L'histoire de l'Algérie dans le domaine de l'assurance se confond, pendant toute la période coloniale, avec l'évolution de l'assurance en France, d'autant plus que les lois antérieures à l'indépendance n'ont été abrogées qu'en 1975, quoi que le législateur ait institué le monopole de l'Etat sur les opérations d'assurances en 1966.
Au lendemain de l'indépendance les opérations d'assurance étaient régies par des textes français, dont la plupart sont devenus caducs avec l'institution du monopole de l'état en 1966, ce n'est qu'en 1975 que fut abrogée toute la réglementation antérieure à l'indépendance. Remplacée en grande partie par la loi n° 80-07 du 9 août 1980.

II.3.2. Structure du marché algérien des assurances : [22]

Les compagnies d'assurances et de réassurance en Algérie sont au nombre de seize, sept sociétés publiques, sept sociétés privées et deux mutuelles.

a) Six sociétés publiques directes :

- ✓ 4 compagnies généralistes opèrent dans toutes les branches d'assurance, la CAAR, la SAA, la CAAT et la CASH, qui représentent ensemble 74 % de la production du marché.
- ✓ 2 compagnies publiques sont spécialisées dans l'assurance du risque crédit : la CAGEX (assurance crédit à l'exportation) et la SGCI (assurance crédit à l'immobilier).

b) Une société publique de réassurance :

- ✓ La CCR, Compagnie centrale de réassurance, bénéficie des cessions préférentielles du marché et de la garantie de l'Etat.

c) Sept sociétés privées :

Elles représentent 20 % de la production globale du marché, acquis en un peu plus de 10 ans, en progression régulière. Ces compagnies sont :

- ✓ CIAR, Compagnie internationale d'assurance et de réassurance.
- ✓ 2A, Algérienne des assurances.
- ✓ TRUST Algeria.
- ✓ GAM, Générale d'assurance méditerranéenne.
- ✓ Salama Assurances (ex Al Baraka Oua Al Amane). • Alliance Assurances.
- ✓ Cardif El Djazaïr.

Pour mémoire, deux sociétés privées, Star Hana (banque BCIA) et Al Rayan (Al Rayan Bank), liées à des groupes bancaires ayant cessé leurs activités bancaires, ont arrêté de ce fait leurs opérations d'assurance.

d) Deux sociétés mutuelles pratiquent l'assurance directe :

- ✓ CNMA, mutuelle agricole, héritière de la mutualité agricole française, représente une part de marché de 6 %.
- ✓ MAATEC, mutuelle des travailleurs de l'éducation nationale et de la culture.

II.3.3. Nouvelles compagnies d'assurance privée étrangères : [23]

- ✓ La Trust Alegria Assurances & Réassurances TRUST avec un capital détenu à 95% par la Trust Real Bahreïn et 5% par Qatar Général insurance qui représente, 2 milliards 50 millions de dinars avec un chiffre d'affaire de 1.827 millions de dinars en 2010.
- ✓ La Compagnie Internationale D'assurance et de Réassurance CIAR détenu par le groupe algérien Soufi avec un capital social représentant 4.167 millions de dinars et un chiffre d'affaire 5.986 millions de dinars en 2010 la plaçant à la première classe des compagnies privées.
- ✓ La Salama Assurances Algérie qui est une filiale du groupe Salama Islamic Arab Insurance Compagny de Dubaï, spécialisé dans les produits d'assurances islamic « TAKAFUL ». avec un capital social de 2 milliards de dinars et un chiffre d'affaire dépassant les deux milliards 600 millions de dinars.

II.4. Les entreprises d'assurance françaises à l'étranger :

Les entreprises françaises qui veulent déployer leur activité d'assurance au sein de l'Espace économique européen bénéficieront de la logique de l'agrément unique – l'agrément délivré par l'ACP vaut « passeport européen » et suffit pour conquérir le marché d'un État membre.
Par contre, si elles souhaitent s'implanter sur un marché hors EEE, il leur faudra se soumettre à la législation locale.

En France, les assurances sont séparées en deux parties. Il y a les assurances vie et les assurances Non vie ou IARD (Incendie, Accidents, Risques Divers). Cependant, ni le code des assurances ni les grands professionnels de l'assurance telle que la FFSA n'arrivent à clairement définir ces deux termes.

En France, les assurances proposées se divisent en plusieurs branches. La plupart de ces branches peuvent se classer dans les deux catégories précédemment citées, mais encore une fois c'est assez flou, mais nous avons fait le tri pour vous. Donc voici à quoi ressemblent les offres fournies et dans quelles catégories elles peuvent être placées :

- **Assurance vie** : l'assurance vie peut comprendre plusieurs offres comme les assurances de risques (c'est à dire décès), les bons de capitalisation et toutes les autres offres d'épargnes.
- **Assurance Non-vie ou IARD** : Assurance de biens et de responsabilités, assurance maladie, assurance accidents corporels, assurance financière (caution, garantie chômage, etc..), assurance juridique (protection juridique) et les offres d'assistances pour toutes sortes de préjudices.

II.5. Principe des compagnies d'assurance en France : [24]

Le principe de l'assurance est simple : l'assureur et l'assuré nouent une relation en signant un contrat par lequel:

- ✓ l'assuré s'engage à verser des primes ponctuelles ou régulières,
- ✓ l'assureur s'engage, lorsqu'un sinistre assuré survient, à verser une indemnisation à l'assuré. L'indemnisation vise à, réparer les dommages matériels et corporels subis.

III. Groupe AXA Assurance :

III.1. Historique : [25]

Depuis ses débuts, le groupe Axa Assurance se veut être l'un des leaders mondiaux sur le marché des produits et service d'assurance.son évolution fut tel le que la société a vite pris une envergure international et s'est vite faite une réputation de choix dans le domaine des offres de produits et services d'assurance.

En une trentaine d'années, le groupe Axa a connu une évolution flagrante dans le domaine de l'assurance et ce grâce, notamment, à Claud Bébéar, qui a activement contribué à cette évolution. En guise de rappel sur le parcours personnel de Claud Bébéar.

Il a commencé comme attaché de direction au sein de l'Ancienne Mutuelle de Rouen durant l'année 1958. C'est en travaillant au sein de celle-ci qu'il a eu l'idée de faire en sorte que la mutuelle en question élargisse son champ d'action sur la scène internationale. Très vite, grâce à son expérience de métier ; l'attaché de direction gravit les échelons et accède à la tête de la Mutuelle il vise ensuite le marchés étrangères en créant une filiale de réassurance aux milieux des années 1970. Rapidement, les filiales se créent, les activités s'élargissent, le changement d'appellation succèdent, pour aboutir finalement à Mutuelles Unies au début des années 1980.Mutuelles Unies rachète le groupe Drouot pour former un nouveau groupe, qui s'affirme de plus en plus dans le secteur des assurances et autres services rattachés. Finalement, c'est en 1985 que le nom groupe Axa voit officiellement le jour, après un long processus de recherche qui donnera à celui-ci pertinence et efficacité dans son domaine d'activité. Depuis ; les activités de groupe se font nombreuses et englobent presque toutes les branches, telles que l'assurance vie, la transmission de patrimoine, les activités bancaires et autres.

III.2. Définition :

Axa est un groupe international français spécialisé dans l'assurance depuis sa création, et dans la gestion d'actifs depuis 1994. En 2012, il est le numéro un de l'assurance dans le monde en termes de chiffre d'affaires.

III.3. Les 4 Principe de groupe AXA : [26]

Le Groupe AXA poursuit son engagement en matière de Responsabilité d'Entreprise par la signature des Principes pour l'Assurance Responsable (Principles for Sustainable Insurance, PSI), lancés officiellement le 19 juin 2012 à Rio de Janeiro, en marge de Rio+20, conférence mondiale sur le développement durable de l'ONU.

Fruits d'un travail de plus de six ans entre l'Initiative Finance du Programme des Nations Unies pour l'Environnement (PNUE FI) et le secteur de l'assurance, ces principes engagent les signataires à intégrer des critères environnementaux, sociaux et de gouvernance au cœur de leur activité d'assureur et dans leurs relations avec leurs parties prenantes :

- ✓ **Principe n°1 :** « Nous intégrerons dans nos prises de décisions les questions environnementales, sociales et de gouvernance liées au secteur de l'assurance. »

- ✓ **Principe n°2 :** « Nous collaborerons avec nos clients et nos partenaires pour les sensibiliser aux questions environnementales, sociales et de gouvernance, les inciter à une meilleure prise en compte du risque et au développement de solutions concrètes. »

- ✓ **Principe n°3 :** « Nous travaillerons aux cotés des gouvernements, régulateurs et autres parties prenantes pour promouvoir une action globale en faveur des questions environnementales, sociales et de gouvernance »

- ✓ **Principe n°4 :** « Nous communiquerons régulièrement et publiquement nos actions en la matière, de façon responsable et transparente. »

III.4. AXA dans le monde :

- ✓ **1ère** Marque mondiale d'assurances pour la seconde année consécutive.
- ✓ **1er** Assureur vie mondial.
- ✓ **4e** Assureur dommage mondial.
- ✓ **6e** Gestionnaire d'actif mondial.
- ✓ Présence : 61 pays.
- ✓ Collaborateurs : 214 000.
- ✓ Clients dans le monde : 95 millions.
- ✓ Chiffre d'affaires : 91 milliards d'euros.
- ✓ Résultat opérationnel : 3,9 milliards d'euros.
- ✓ Résultat net : 2,7 milliards d'euros.
- ✓ Ratio de Solvabilité I : 182 % en hausse de 11 points.
- ✓ Dividende par action : 0,69 euro en hausse de 25 % (sous réserve de l'approbation de l'assemblée générale du 27 avril 2011).

III.5. AXA en Algérie : [27]

A travers notre implantation en Algérie, notre souhait est de construire une présence de long terme, ainsi l'objectif du Groupe AXA est de contribuer en Algérie, comme dans les autres pays dans lesquels il est présent, à une croissance durable de l'économie et au bien-être de la société par la prise en charge des risques auxquels font face les individus et les acteurs économiques.

Dans le respect de la réglementation applicable en matière d'assurance, nous avons opté pour un partenariat privilégié et un ancrage solide en s'associant avec la Banque Extérieure d'Algérie (BEA) et le Fonds National d'Investissement (FNI), deux partenaires publics essentiels de l'économie algérienne qui partagent le même objectif de croissance économique. Notre stratégie est de nous positionner en Algérie comme un assureur généraliste, où les activités seront développées aussi bien sur le marché de l'assurance dommage que sur celui de l'assurance de personnes.

III.6. Des activités menées par deux compagnies d'assurances Axa : [28]

AXA Assurances Algérie Dommage dotée d'un capital de 2 milliards de dinars et AXA Assurances Algérie Vie dotée d'un capital d'1 milliard dinas.

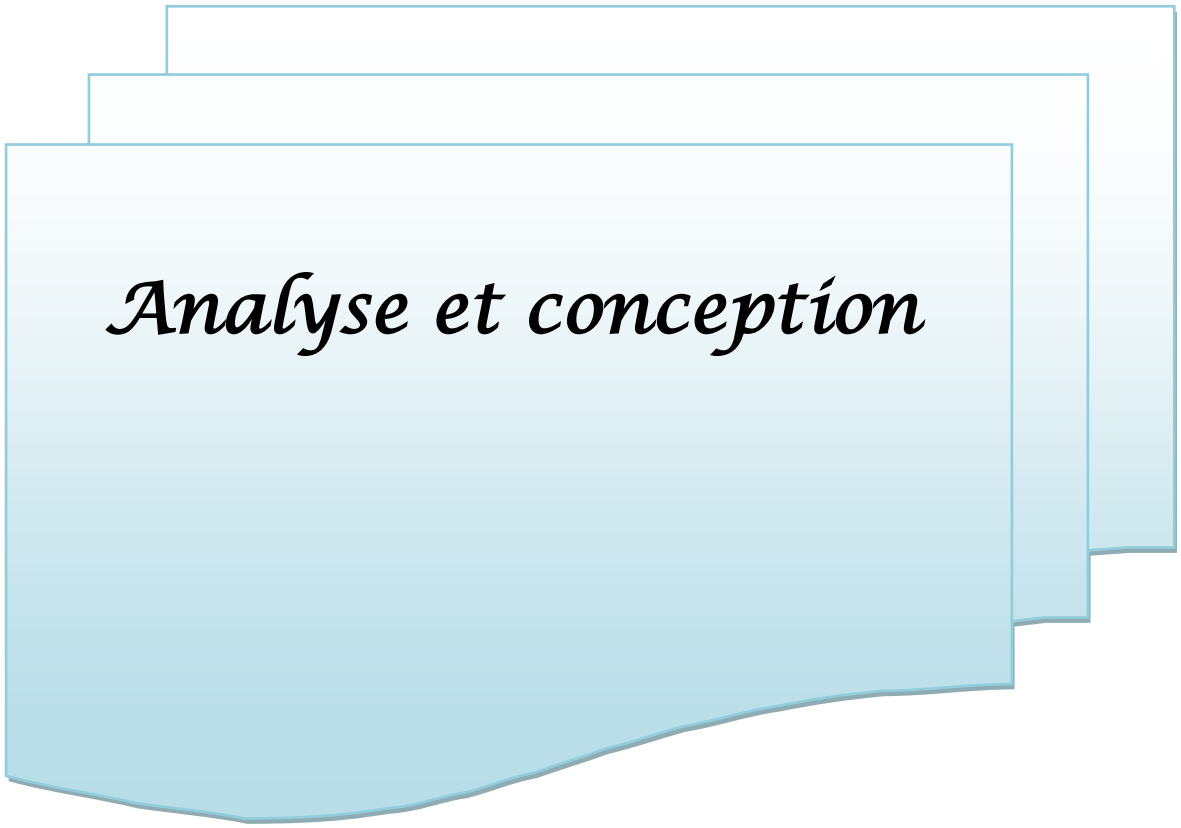
Notre ambition est de devenir la société préférée de nos clients. L'Algérie représente un marché à fort potentiel dans lequel AXA compte se différencier par:

- Des produits d'assurance innovants conçus pour répondre aux besoins de nos différents types de clients.
- La meilleure qualité de service à tous les niveaux, garantie par le respect des standards d'AXA.
- La capacité d'AXA à répondre aux attentes des clients par des preuves réelles et tangibles, afin d'établir une véritable relation de confiance.

IV. Conclusion :

Dans ce chapitre, nous avons présenté le concept des assurances en Algérie et à l'étranger, cela pour que vous ayez la possibilité de bien vous situer dans ce vaste domaine et bien connaitre ses différents types en incluant le contrat et les sociétés d'assurances en particulier le groupe AXA qui est l'objectif de notre application. Le chapitre suivant sera consacrer a l'étape Analyse et conception.

CHAPITRE III:



I. Introduction :

Avant d'entamer la phase de conception d'une façon précise et détaillée, il faudra commencer par une exploration profonde nous permettant de refléter le futur système avant sa concrétisation. Pour se faire, nous avons Opté pour une démarche de conception orientée objet (XP, eXtreme Programming), Ces méthodes se veulent plus pragmatiques que leurs homologues classiques et commencent à être appliquées dans de multiples domaines. On utilisera le langage UML comme langage de modélisation, ce qui nous permettra de mieux représenter l'aspect statique et dynamique de notre application grâce aux diagrammes qu'il offre.

II. Problématique :

Le point le plus important dans notre application est de satisfaire les besoins du client au moindre coût et dans les moindres délais, pour cela on a opté à utiliser la méthodologie agile XP qui fournit une souplesse et une visibilité en impliquant le client du début à la fin du projet et en adaptant un processus de développement itératif et incrémental.

III. UML est-il soluble dans les méthodes agiles ? [29]

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue. UML unifie à la fois les notations et les concepts orientés objet. Il ne s'agit pas d'une simple notation graphique, car les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage.

III.1. UML et XP : [29]

D'après le second principe du manifeste agile, l'utilisation de modèles UML à des fins de documentation doit être restreinte au strict nécessaire … Regardons cependant plus en détail ce qu'en dit vraiment eXtreme Programming, et ses promoteurs de la modélisation.

- ✓ XP n'interdit pas les documents, mais pose dessus un regard critique …
- ✓ XP n'interdit pas la modélisation, mais rappelle que seul un code opérationnel est garant de qualité…
- ✓ Si la création d'un modèle est une demande Client, alors il s'agit d'une « user story » à intégrer dans une ou plusieurs itérations.

Donc on peut utiliser sans problème UML dans XP, mais comment ?

III.2. Utilisation d'UML dans XP : [29]

- À la place des « user stories » ?

Un ou plusieurs diagrammes, selon les besoins, parmi les diagrammes suivants :

❖ Diagrammes des cas d'utilisation :

Permettent d'identifier les fonctionnalités fournies par le système (cas d'utilisation), les utilisateurs qui interagissent avec le système (acteurs), et les interactions entre ces derniers. Les cas d'utilisation sont utilisés dans la phase d'analyse pour définir les besoins de "haut niveau" du système. Les objectifs principaux des diagrammes des cas d'utilisation sont:

-fournir une vue de haut-niveau de ce que fait le système.
- Identifier les utilisateurs ("acteurs") du système.
- Déterminer des secteurs nécessitant des interfaces homme-machine.

En fait, des descriptions textuelles des cas d'utilisation sont souvent employées pour compléter ces derniers et représenter leurs fonctionnalités plus en détail.

❖ ·Diagrammes d'interaction :

Les diagrammes d'interaction ou dynamiques (*Interaction Diagram*) rassemblent :

- ✓ *Diagramme de séquence* (*Sequence Diagram*) : représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
- ✓ *Diagramme de communication* (*Communication Diagram*) : représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.

- À la place des CRC Cards ?

Un ou plusieurs diagrammes, selon les besoins, parmi les diagrammes suivants :

❖ Diagramme de classes :

Le diagramme de classes est un modèle permettant de décrire de manière statique, abstraite et générale les liens et relations entre objets. Une classe permet de décrire un ensemble d'objets (attributs et comportement), tandis qu'une relation ou association permet de faire apparaître des liens entre ces objets. On peut donc dire :

- un objet est une instance de classe.
- un lien est une instance de relation.

IV. Présentation d'UML :

1. Définition :

UML est un langage pour documenter et spécifier graphiquement tous les aspects d'un système logiciel, il est organisé autour de notations, schéma, diagrammes et autres symboles qui permettent une certaine abstraction du système à modéliser en offrant des vues du futur système.

2. Modélisation avec UML :

UML permet de représenter des modèles, mais il ne définit pas d'élaboration de modèles.

Les autres auteures d'UML conseillent tout de même une démarche pour favoriser la réussite d'un projet, cette démarche doit être :

a) Une démarche itérative et incrémentale : Pour comprendre et représenter un système complexe, pour analyser les étapes, pour favoriser le prototype et pour réduire et maîtriser l'inconnu.

b) Une démarche guidée par les besoin des utilisateurs : tout est basé sur le besoin des utilisateurs du système, le but du développement lui-même est de répondre a leurs besoins.

Chaque étape sera affinée et validée en fonction des besoins des utilisateurs.

c) Une démarche centrée sur l'architecture logicielle : C'est la clé de voute du succès d'un développement, les choix stratégiques définiront la qualité du logiciel.

3. La démarche de modélisation avec L'UML :

La démarche de modélisation choisie pour concevoir notre application peut être représentée graphiquement comme suite :

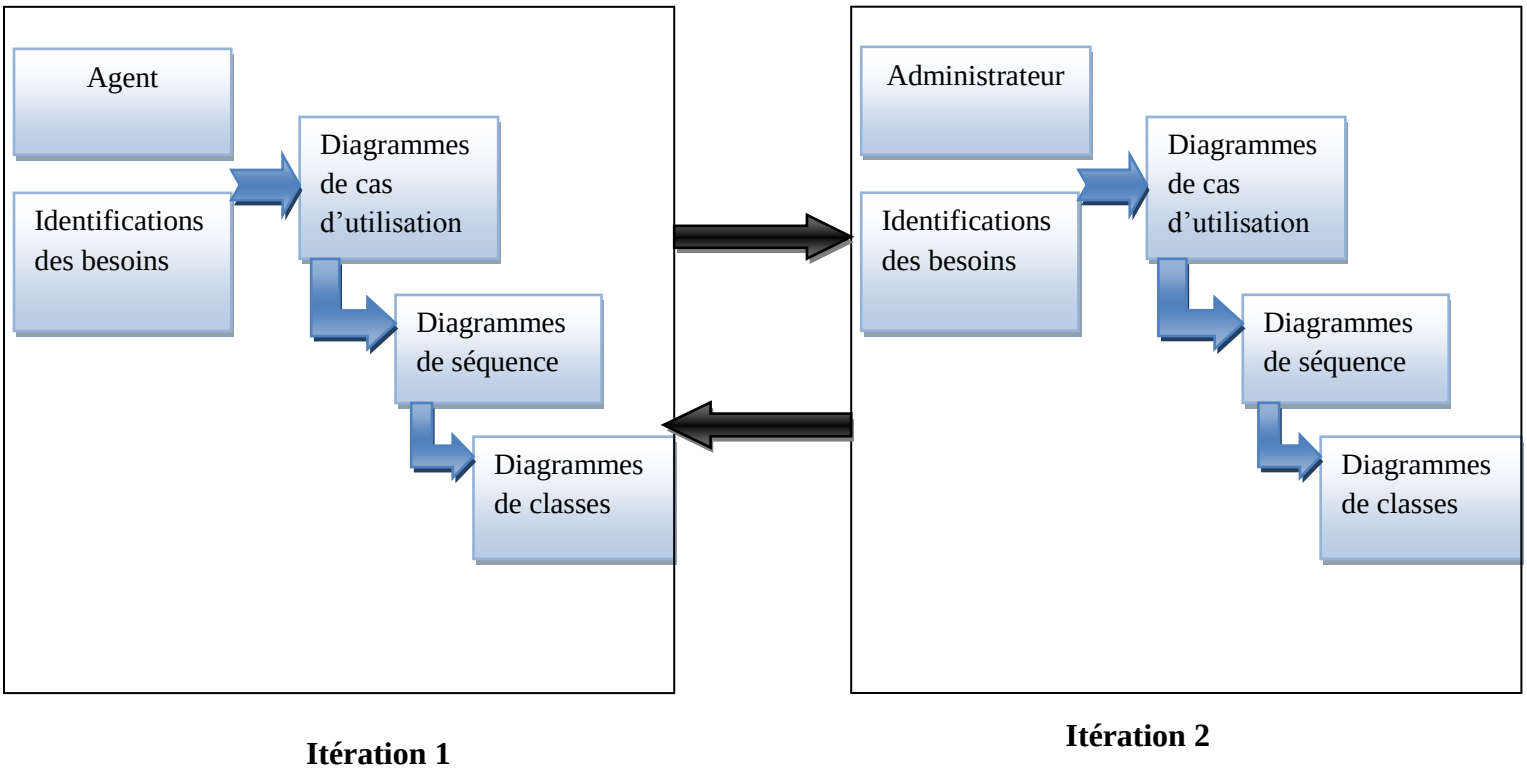


Figure III.1 : La démarche de modélisation de l'application.

IV .1. Spécification des besoins & cas d'utilisation :

1. Définitions de base :

- **Acteur :** un acteur représente un rôle joué par une personne ou une chose qui interagit avec un système. Les acteurs se déterminent en observant les utilisateurs directs du système, ceux qui sont responsables de son exploitation ou de sa maintenance, ainsi que les autres systèmes qui interagissent avec le système en question.
- **Tâche :** c'est l'ensemble des fonctions qu'un acteur bien spécifié peut effectuer.
- **Scénario :** c'est une succession particulière d'enchainement, s'exécutant du début à la fin du cas d'utilisation.

2. Identification des acteurs :

Un acteur est une entité externe (personne, machine, autre système, etc.) interagissant avec le système et qui n'appartient pas a se dernier. Les acteurs se déterminent en observant les utilisateurs directs du système, ceux qui sont responsables de son exploitation ou de sa maintenance, ainsi que les autres systèmes qui interagissent avec le système en question.
Les acteurs de notre système sont :

- **Agent :** Toute personne qui gère les clients, les contrats.
- **Administrateur** : C'est la personne chargé d'administrer le site c.à.d. la gestion des Agents et leurs profiles, …
- **Client**: Toute personne qui se présente a la société pour satisfaire un besoin.

IV.2. définition des itérations :

Le critère de définition des itérations se base sur les deux points suivants :

1. Ressemblance entre cas d'utilisation
2. Dépendance entre cas d'utilisation

IV.2.1.réalisation de l'itération 1 :

1. Identification des cas d'utilisations : [30]

Un cas d'utilisation est utilise pour définir le comportement d'un système ou la sémantique de toute autre entité sans révéler sa structure interne. Chaque cas d'utilisation spécifie une séquence d'action, y compris des variantes, que l'entité réalise, en interagissant avec les acteurs de l'entité. La responsabilité d'un cas d'utilisation est de spécifier un ensemble d'instances, ou une instance de cas d'utilisation représente une séquence d'actions que le système réalise et qui fournit un résultat observable par l'acteur.

2. Spécification des tâches :

Pour chaque acteur, nous spécifions les taches qu''il assure.

Acteur	Tâche
Agent	T1 : S'authentifier pour accéder a son espace T2 : Gérer des clients (créer, modifier, Consulter) T3 : Gestion des contrats (créer, modifier, Consulter) T4 : Se déconnecter

3. Spécification des scenarios :

Chacune de tache effectuée par un acteur est décrite par un ensemble de scénarios, ces scénarios sont illustrés dans le tableau ci-dessous.

Acteurs	Tâches	Scénarios
Agent	T1:S'authentifier.	S1: Saisir login et mot de passe.
	T2: Gestion des clients	S2: Consulter la liste des clients. S3 : Créer un client. S4: Modifier un client.
	T3: Gestion des contrats	S5: Consulter la liste des contrats. S6: créer un contrat. S7: Modifier un contrat.
	T4 : Se déconnecter	S8 : cliquer sur le bouton déconnecter

4. Spécification des cas d'utilisation :

Les figures qui suivent représentent une description de certains cas d'utilisation de l'itération 1:

- ✓ **Cas d'utilisation « S'authentifier »:**

Use case: S'authentifier.
Scenarios: S1.
Rôle : Agent.
Description :

1. Le système affiche la page d'accueil spécifiée.
2. L'Agent saisit son **login** et **mot de passe** puis clique sur le bouton **connexion**.
3. Le système affiche l'espace personnel de l'agent.

Figure III .2: Cas d'utilisation « S'authentifier ».

✓ **Cas d'utilisation « Gestion des Clients »:**

Use case: Gestion des clients.
Scenarios: S1, S2, S3, S4.
Rôle : L'agent
Description :

1. S'authentifier pour accéder a son espace.
2. Le système affiche l'interface Agent.
3. Sélectionner un lien parmi les liens correspondant à la gestion des clients (ajouter. Modifier).
5. Le système affiche la page correspondante.
6. Effectuer des modifications sur la table « Client » de la base de données Puis valider.

Figure III.3 : Cas d'utilisation « Gestion des clients ».

✓ **Cas d'utilisation « Gestion des Contrats » :**

Use case: Gestion des contrats.

Scenarios: S1, S5, S6, S7.

Rôle : Agent.

Description :

1. S'authentifier pour accéder a son espace.
2. Le système affiche l'interface Agent.
3. Sélectionner un lien parmi les liens correspondant à la gestion des contrats.(Ajouter. Modifier, …).
4. Le système affiche la page correspondante.
5. Effectuer des modifications sur la table « Contrat» de la base de données puis valider.

Figure III.4 : Cas d'utilisation « Gestion des contrats »

5. Digramme de cas d'utilisation : [31]

Un diagramme de cas d'utilisation : un cas d'utilisation (use case) modélise une interaction entre le système informatique à développer et un utilisateur ou acteur interagissant avec le système. Plus précisément, un cas d'utilisation décrite une séquence d'actions réalisées par le système qui produit un résultat observable pour un acteur.

❖ **Diagramme de cas utilisation de l'itération 1 :**

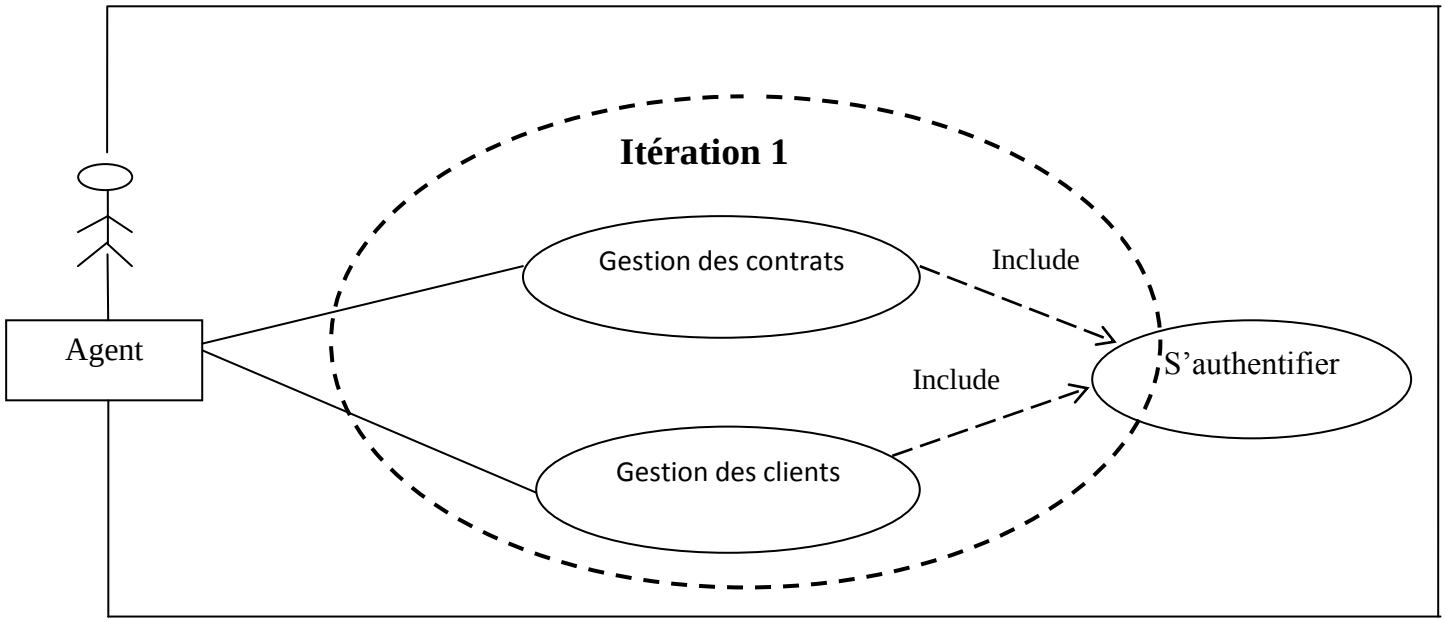


Figure III.5 : Diagramme du cas d'utilisation de l'itération 1.

6. Diagrammes de séquence pour itération 1: [31]

Les diagrammes de séquence présentent la coopération entre différents objets. Les objets sont définis et leur coopération est représentée par une séquence de messages entre eux.
Apres avoir déterminé nos besoins et réaliser les diagrammes des cas d'utilisations nous allons élaborer quelques diagrammes de séquences de l'itération 1.

✓ **Diagramme de séquence du cas d'utilisation « Ajouter un client » :**

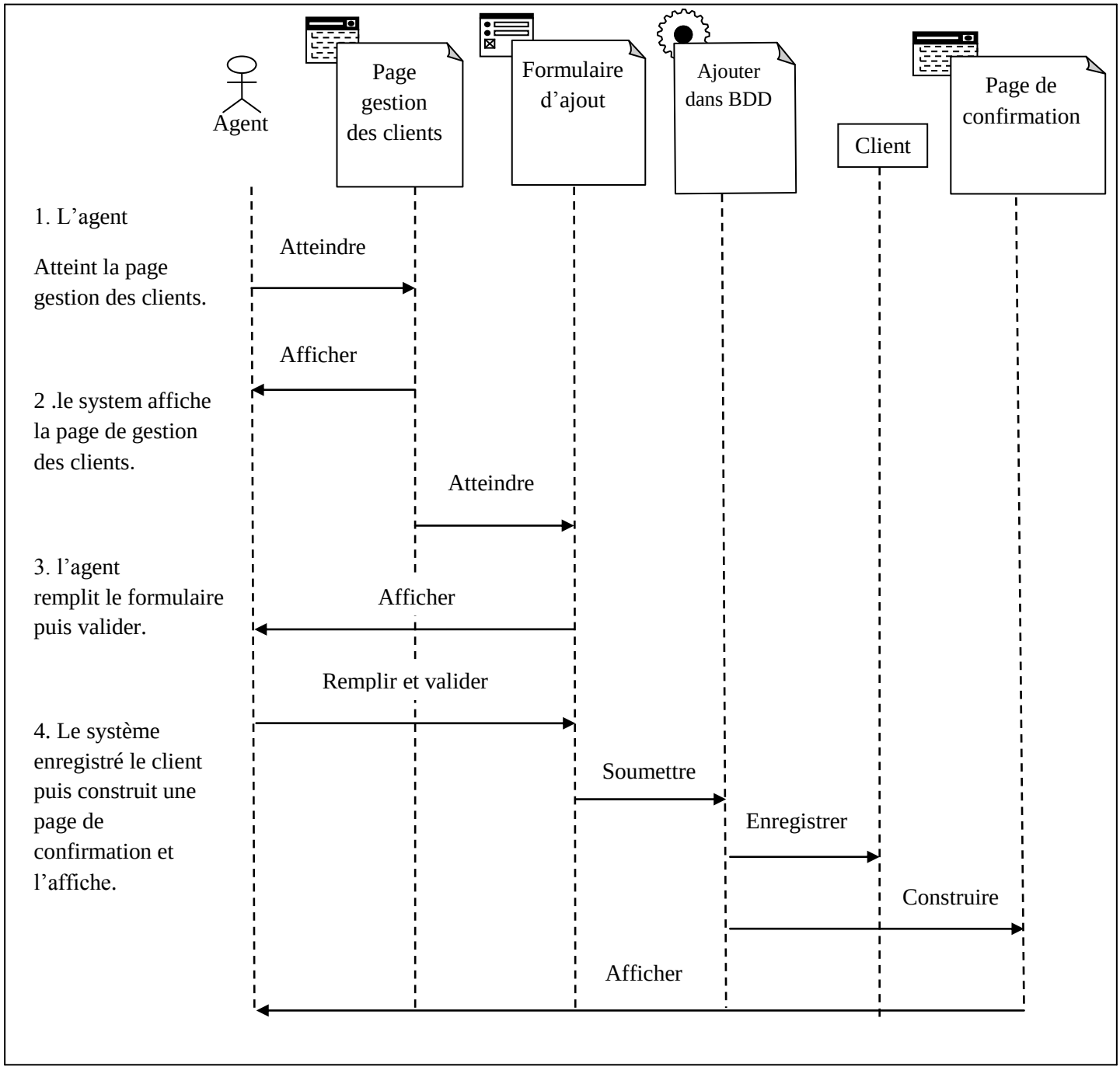


Figure III.6 : Diagramme de séquence du cas d'utilisation « Ajouter un client ».

✓ **Diagramme de séquence du cas d'utilisation « Créer un contrat» :**

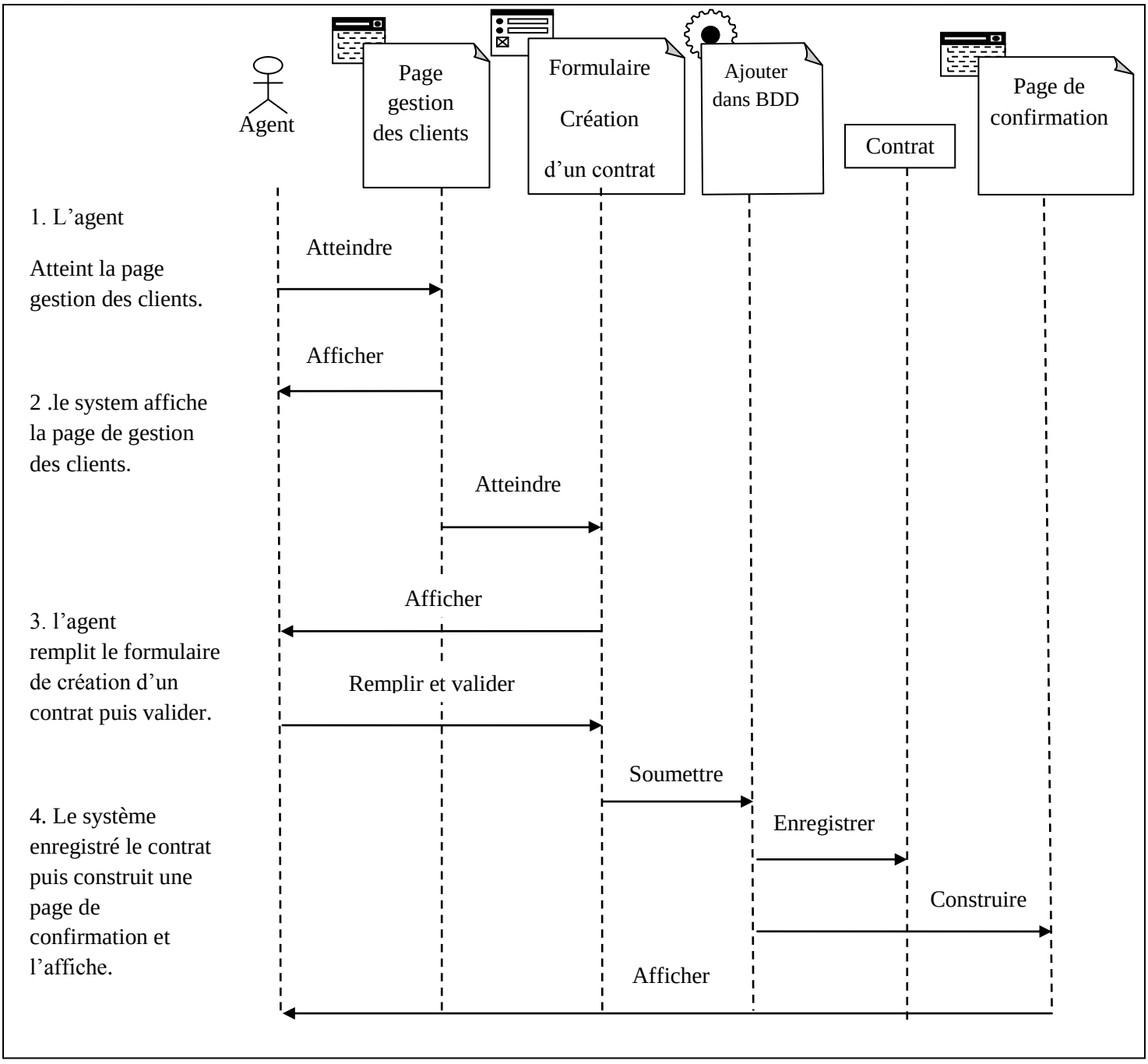


Figure III.7 : Diagramme de séquence du cas d'utilisation « Créer un contrat ».

7. Diagrammes de classe généraux pour l'itération 1 : [32]

Les diagrammes de classes sont sans doute les diagrammes les plus utilisés d'UML. Ils décrivent les types des objets qui composent un système et les différents types de relations statiques qui existent entre eux. Ils font abstraction du comportement du système.

Une fois que les diagrammes de séquence sont élaborés, on passera aux diagrammes de classes qui représentent la vue logique des objets pages.

✓ Diagramme de classe général du cas d'utilisation « Ajouter un Client» :

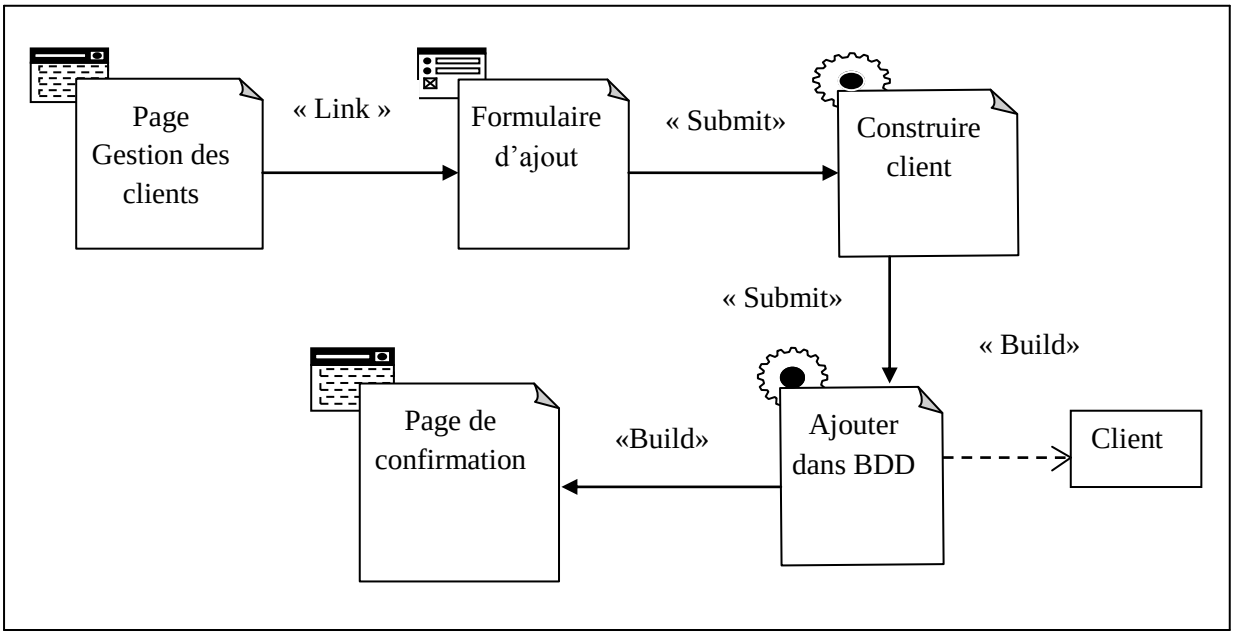


Figure III.8 : Diagramme de classe général du cas d'utilisation « Ajouter un Client».

✓ **Diagramme de classe général du cas d'utilisation « Création un Contrat» :**

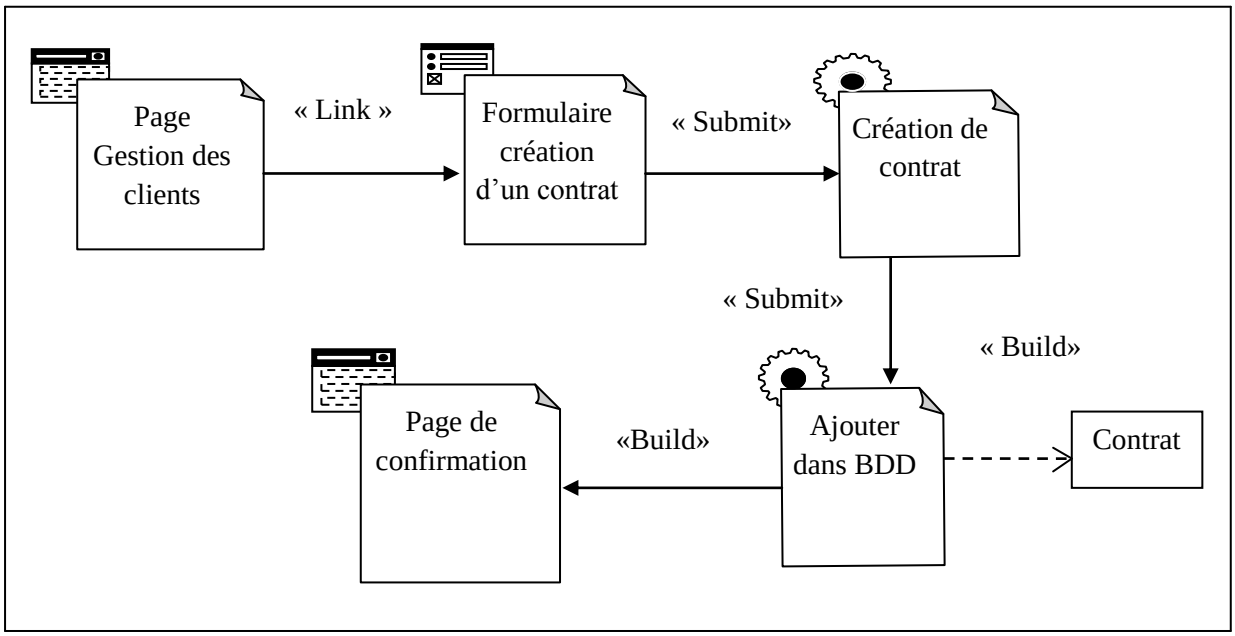


Figure III.9 : Diagramme de classe général du cas d'utilisation « Création un contrat».

8. Diagrammes de classe pour l'itération 1 :

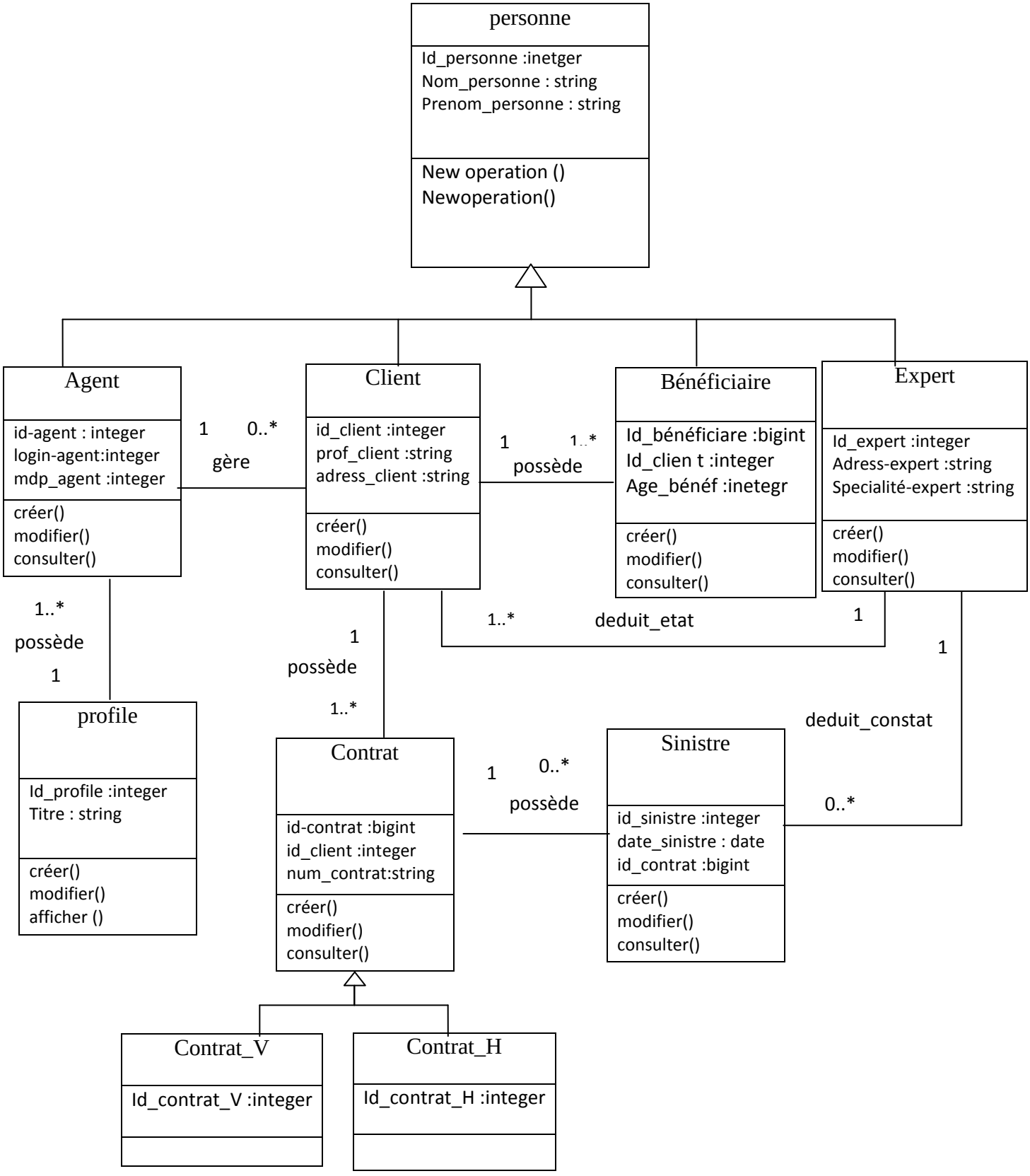


Figure.III.10. : Diagramme de classe pour l'itération 1.

IV.2.2.réalisation de l'itération 2 :

1. **Spécifications des taches :**

Acteur	Tâches
Administrateur	T1 : S'authentifier pour accéder a son espace T2: Accéder à l'espace administrateur T3 : Gestion des agents (créer, modifier, consulter) T4 : Gestion des profiles (créer, modifier, consulter) T5: Se déconnecté

2. **Spécification des scenarios :**

Acteur	Tâches	Scénarios
Administrateur	T1 : S'authentifier	S1 : Saisir login et mot de passe.
	T2: Gestion des agents.	S2: Consulter la liste des agents. S3: Créer un agent. S4: Modifier un agent.
	T3: Gestion des profiles.	S5 : Consulter la liste des profiles des agents. S6 : créer un profile. S7 : Modifier un profile.
	T4: Se déconnecter.	. S8 : Cliquer sur le bouton déconnexion.

3. Spécification des cas d'utilisation :

✓ **Cas d'utilisation « S'authentifier »:**

Use case: S'authentifier.

Scenarios: S1.

Rôle : Administrateur

Description :

1. Le système affiche la page d'accueil spécifiée.
2. L'Administrateur saisit son **login** et **mot de passe** puis clique sur le bouton **connexion**.
3. Le système affiche l'espace personnel de l'administrateur.

Figure III .11: Cas d'utilisation « S'authentifier »

✓ **Cas d'utilisation « Gestion des Agents »:**

Use case: Gestion des Agents.
Scenarios : S1, S2, S3, S4,
Rôle : Administrateur.
Description :

1. Saisir le login et le mot de passe puis valider.
2. Sélectionner un lien parmi les liens correspondant à la gestion des agents (modifier, ajouter…).
3. Le système affiche la page correspondante.
4. Effectuer des modifications sur la table « Agent » de la base de données puis valider.

Figure III.13: Cas d'utilisation « Gestion des agents ».

4. **Diagramme de cas utilisation pour l'itération 2 :**

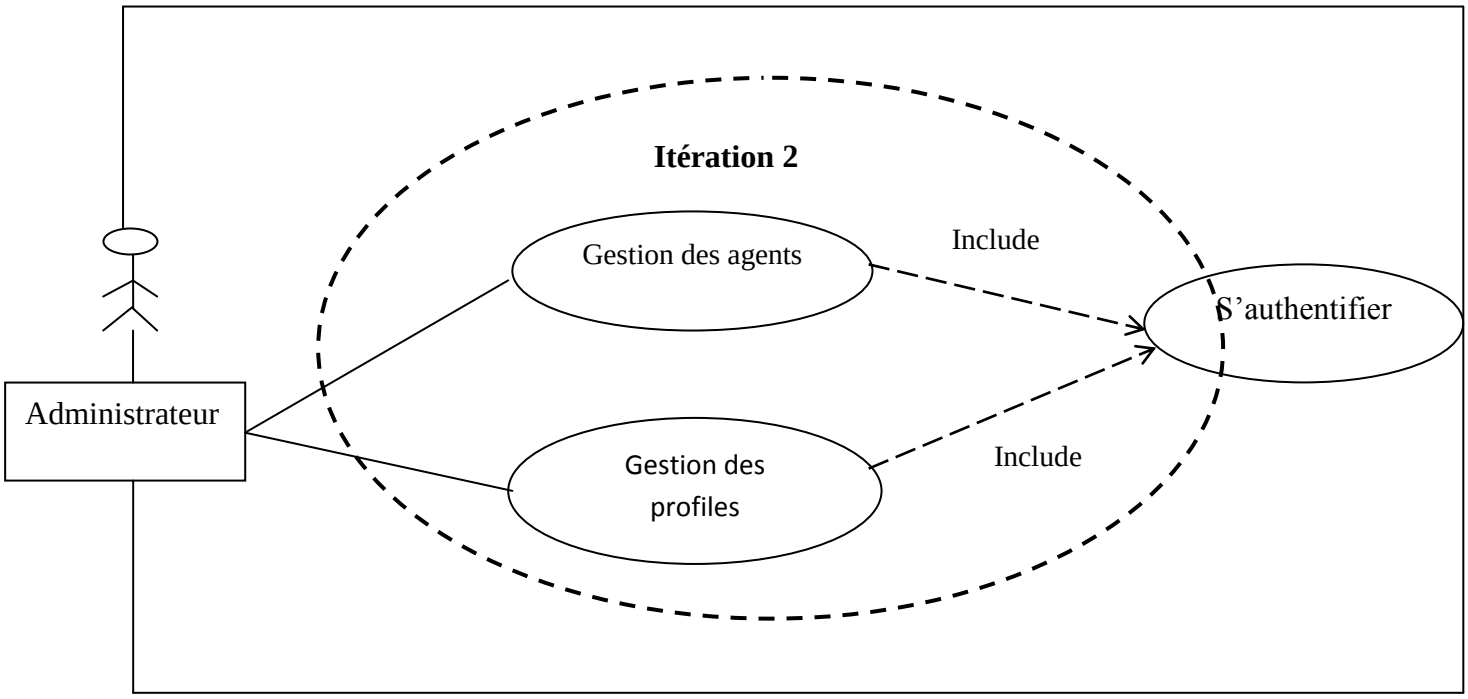


Figure III.14 : Diagramme du cas d'utilisation pour l'itération 1.

5. **Diagrammes de séquence pour itération 2:** quelques diagrammes de séquences de l'itération 2.

✓ **Diagramme de classe général du cas d'utilisation « Authentification» :**

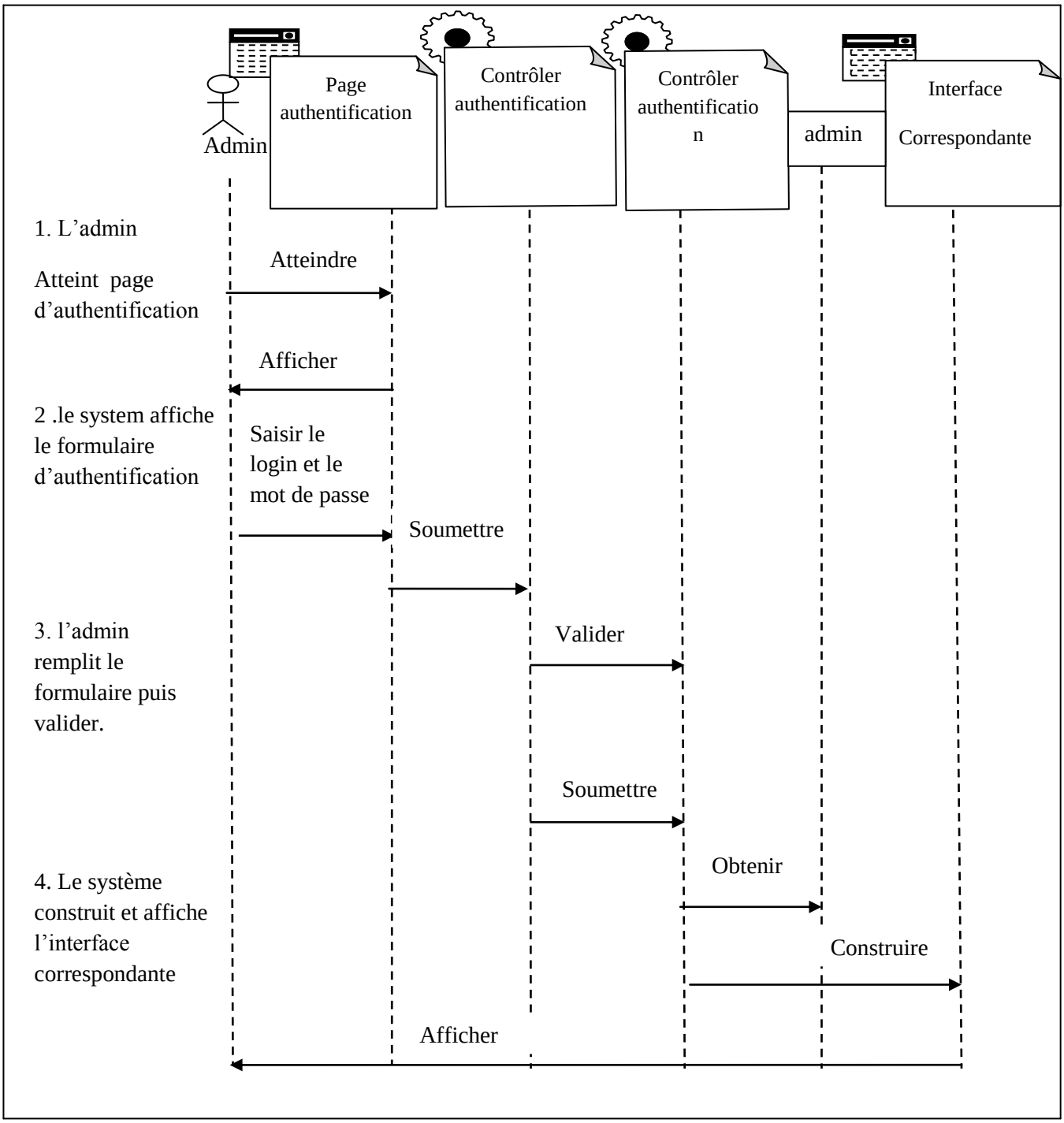


Figure III.15 : Diagramme de séquence du cas d'utilisation « Authentifier ».

✓ **Diagramme de séquence du cas d'utilisation « Ajouter un agent» :**

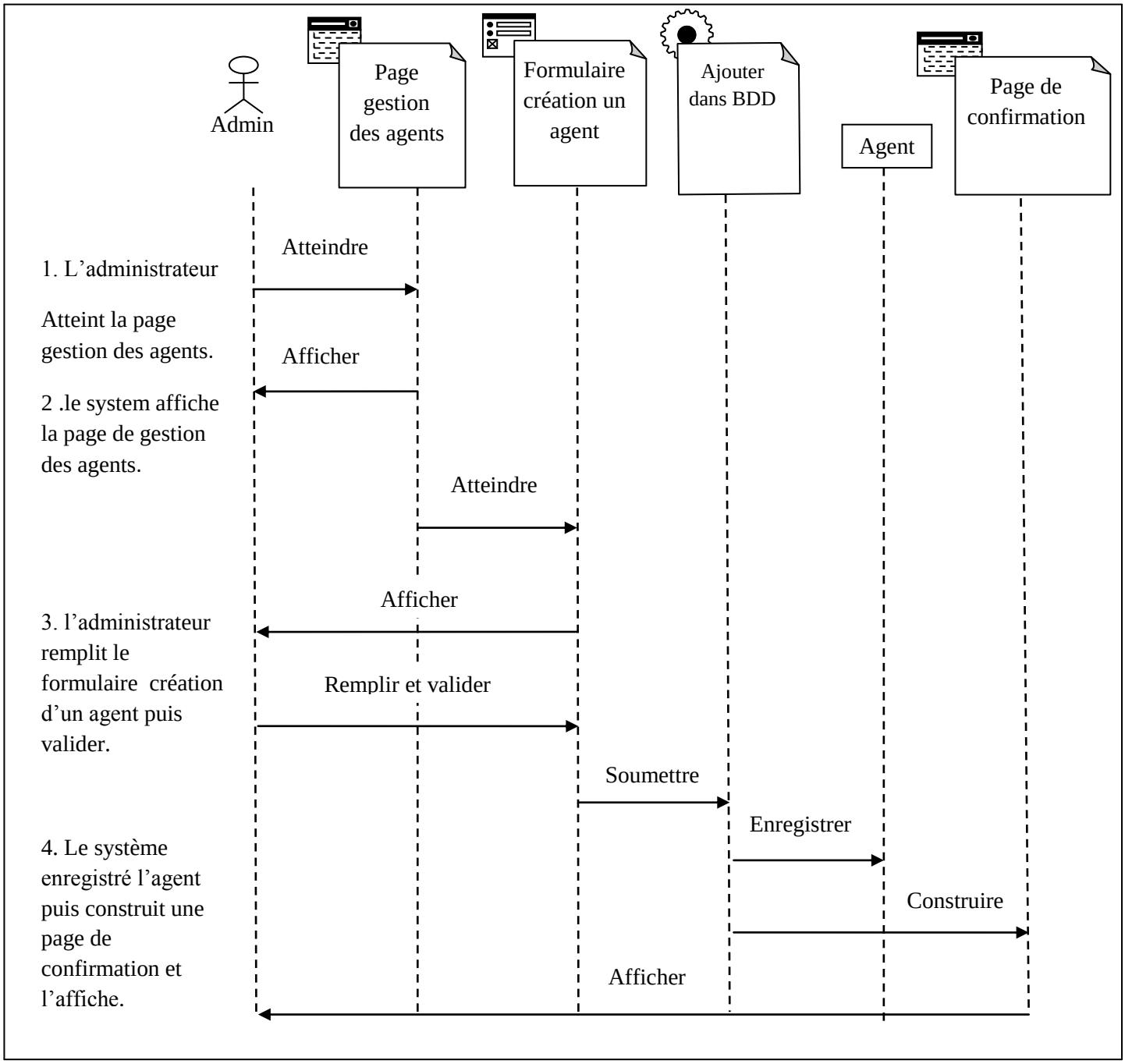


Figure III.16 : Diagramme de séquence du cas d'utilisation « Ajouter un agent ».

6. Diagrammes de classe généraux pour l'itération 2 :

✓ Diagramme de classe général du cas d'utilisation « Authentification» :

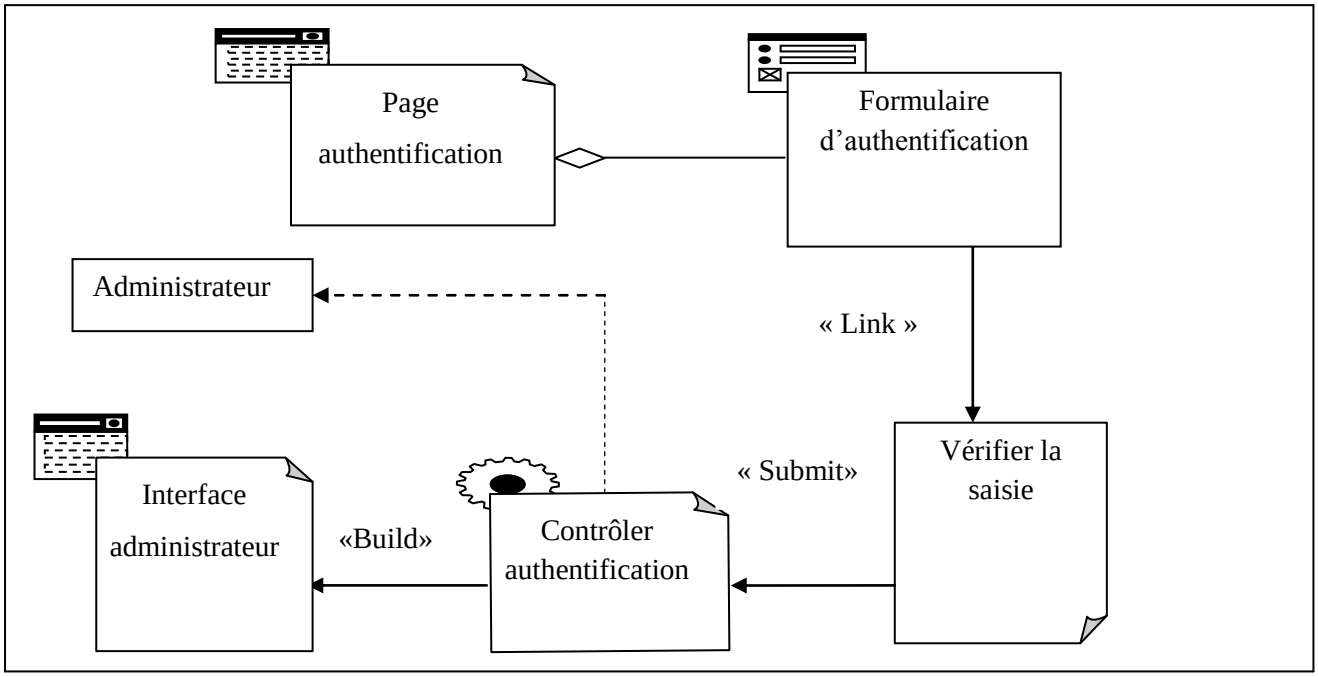


Figure III.17 : Diagramme de classe général du cas d'utilisation «Authentification »

✓ Diagramme de classe général du cas d'utilisation « Ajouter un agent» :

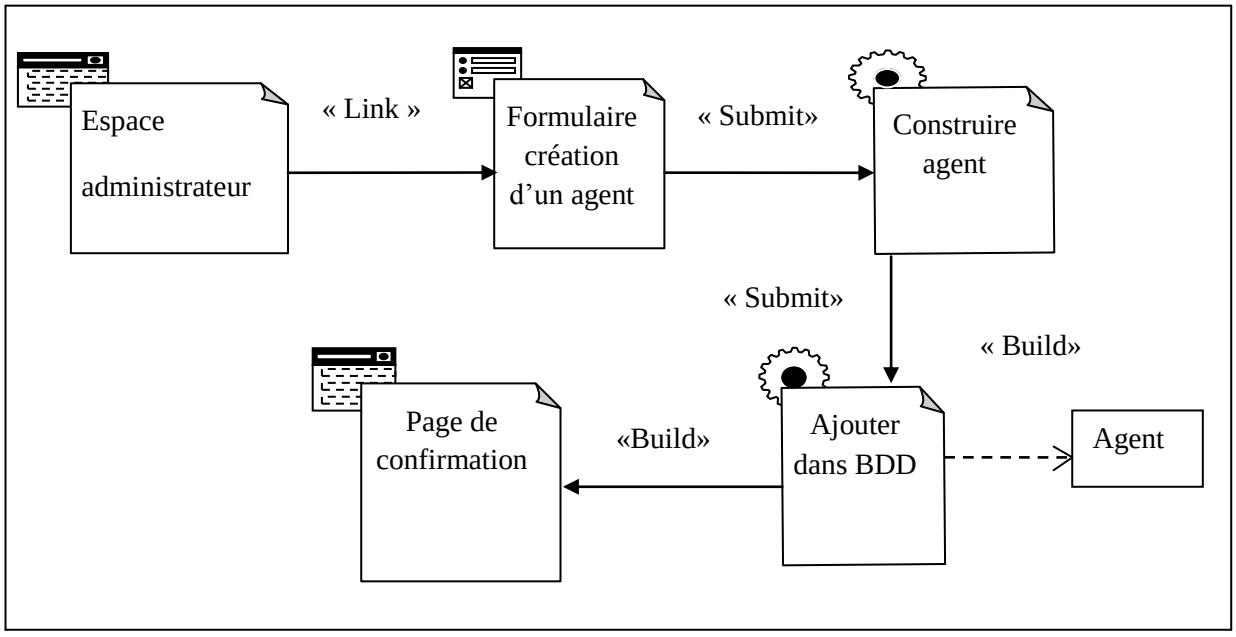


Figure III.18 : Diagramme de classe général du cas d'utilisation « Ajouter un agent».

7. Diagrammes de classe pour l'itération 2 :

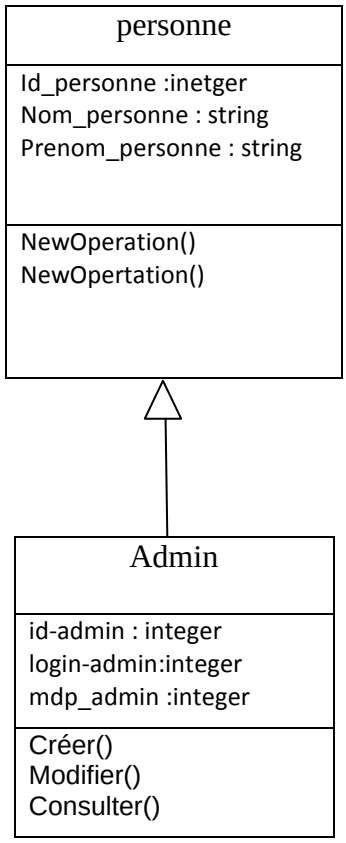


Figure.III.19. Diagramme de classe pour l'itération 2**.**

V. Diagramme de cas d'utilisation global :

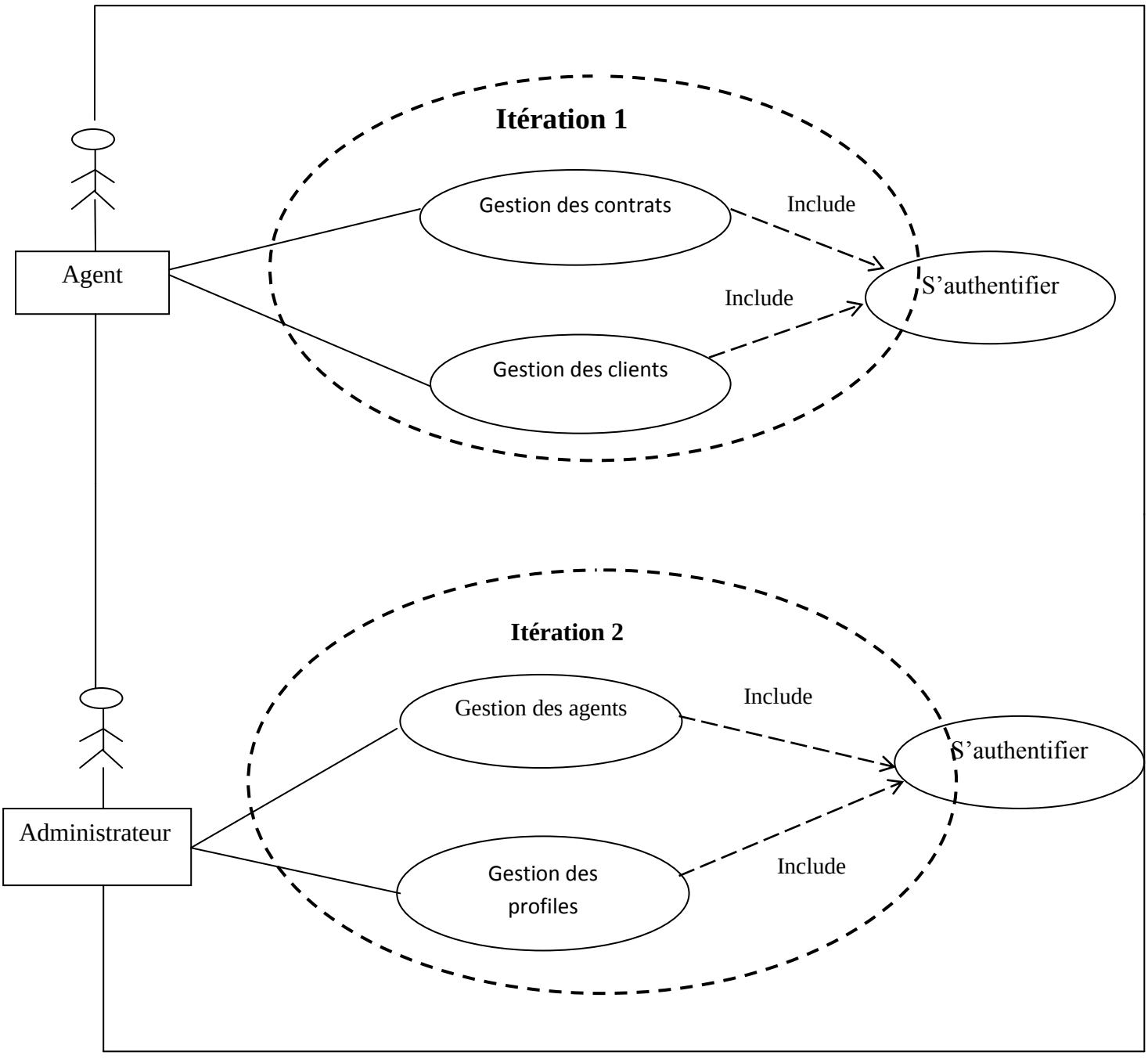


Figure.III.20. Diagramme de cas d'utilisation global.

VI. Diagramme de classe global :

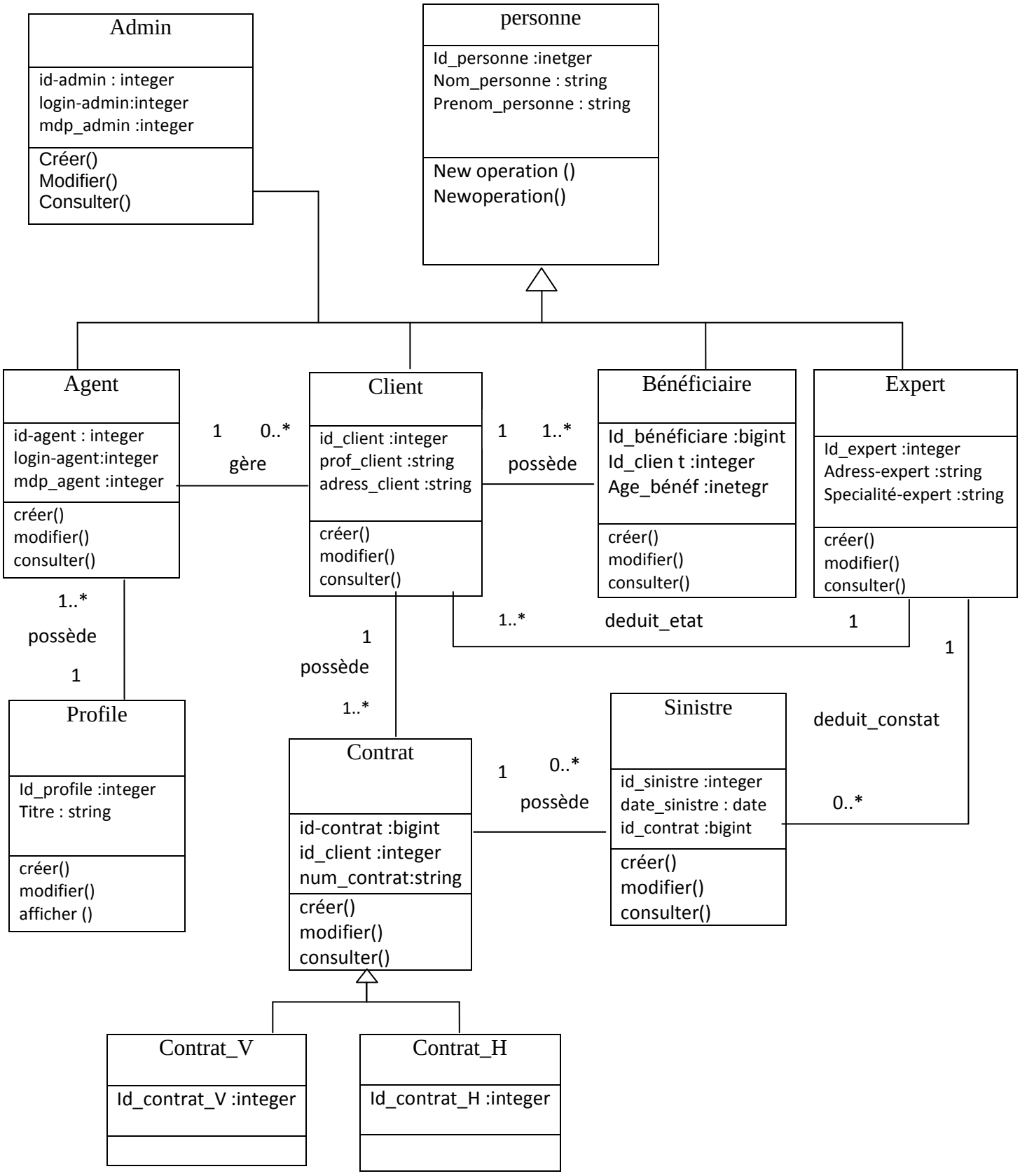


Figure III.21 : Diagramme de classe global.

VI.1.Le modèle relationnel :

Administrateur (id_admin,login_admin ,mdp_admin).

Personne (Id_personne ;nom_personne ,prenom_personne).

Agent (id_agent,id_personne* , login_agent, mdp_agent).

Client (id_client, id_personne*,profession_client,téléphone_client,adresse_client),

Contrat(id_contrat,id_client*,,num_contrat,montant_paye,type_contrat,dsignation_contrat, datee_contrat,etat_contrat,date_dec,taux_remboursement,transfert_beneficaire,date_hosp,

validite, type_hosp,date_reactivation),

Sinistre (id_sinistre, designation,date_sinistre,type_sinistre,id_contrat*) .

Bénéficaire (id_beneficiaire, id_client* ,id_personne*, ,age_benif).

Expert (id_expert, id_personne*,adresse_expert,telephone_expert,specialite_expert).

Profile (Id_profile.titre_profile).

VII. Conclusion :

Dans ce chapitre nous avons utilisé la méthode agile XP pour l'analyse et la conception de notre application en utilisant le langage de modélisation UML. Dans le chapitre suivant, qui touche à la réalisation, nous essayons de construire notre application par itération.

CHAPITRE IV :

La Réalisation

I. Introduction :

On s'intéressera dans ce chapitre à l'implémentation de notre future application et les différentes fonctionnalités qu'elle nous offre ainsi qu'aux différents outils que nous avons utilisés lors de sa réalisation. En effet, nous y verrons les langages utilisés ainsi que les différentes technologies utilisées pour son développement (Système d'exploitation, outils de conception web, IDE etc.…), et nous aborderons ensuite son fonctionnement et ses différentes interfaces.

I.1. Application web : [33]

1).Définition :

En informatique, une application Web (aussi appelée WebApp) est un logiciel applicatif manipulable grâce à un navigateur Web. De la même manière que les sites Web, une WebApp est généralement placée sur un serveur et se manipule en actionnant des widgets à l'aide d'un navigateur Web, via un réseau informatique (Internet, intranet, réseau local, etc.).

Les messageries web, les systèmes de gestion de contenu, les blogs sont des applications Web. Les moteurs de recherches, les logiciels de commerce électronique, les jeux en ligne, les logiciels de forum peuvent être sous forme d'application Web.

Des appareils réseau tels que les routeurs sont parfois équipés d'une application Web dans leur micro-logiciel.

Les applications Web font partie de l'évolution des usages et de la technologie du Web.

II.L'architecture client/serveur : [34]

II.1. Le Client-serveur :

C'est un modèle informatique basé sur le traitement distribué selon lequel un utilisateur lance un logiciel client à partir d'un ordinateur relié à un réseau, déclenchant simultanément le lancement d'un logiciel serveur situé dans un autre ordinateur possédant les ressources souhaitées par l'utilisateur.

II.2. Présentation de l'architecture d'un système client/serveur : [34]

De nombreuses applications fonctionnent selon un environnement client/serveur, cela signifie que des machines clientes (des machines faisant partie du réseau) contactent un serveur, une machine généralement très puissante en termes de capacité d'entrée-sortie, qui leur fournit des services.

Les services sont exploités par des programmes, appelés programmes clients, s'exécutant sur les machines clientes. On parle ainsi de client FTP, client de messagerie ..., lorsque l'on désigne un programme, tournant sur une machine cliente, capable de traiter des informations qu'il récupère auprès du serveur. (Dans le cas du client FTP il s'agit de fichiers, tandis que pour le client messagerie il s'agit de courriers électroniques). Dans un environnement purement Client/serveur, les ordinateurs du réseau (les clients) ne peuvent voir que le serveur, c'est un des principaux atouts de ce modèle (chaque machine est soit client, soit serveur).

II.3. Notions de bases :

- **Client :** C'est le processus demandant l'exécution d'une opération à un autre processus par envoi d'un message contenant le descriptif de l'opération à exécuter et attendant la réponse à cette opération par un message en retour.

- **Serveur :** C'est un processus accomplissant une opération sur demande d'un client.

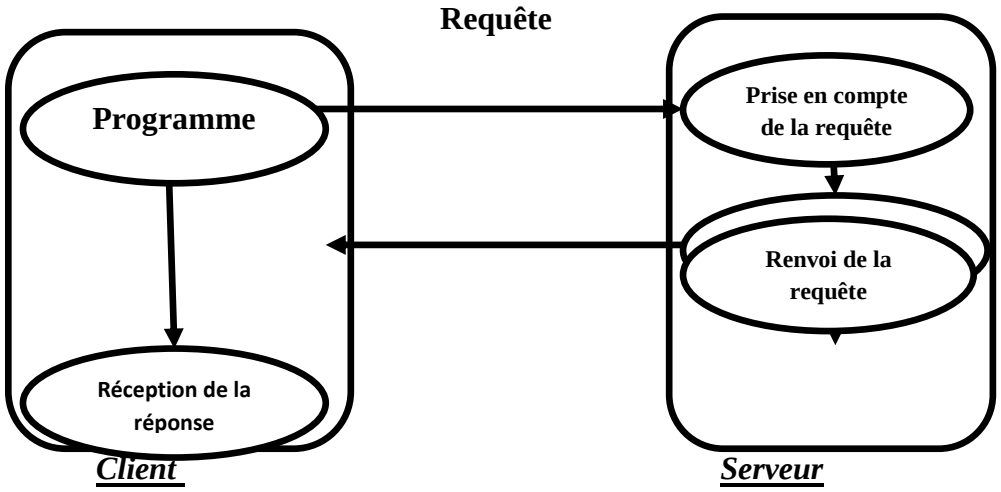
- **Requête :** C'est un message transmis par un client à un serveur décrivant l'opération à exécuter pour le compte d'un client.

- **Réponse :** C'est un message transmis par un serveur à un client suite à l'exécution d'une opération contenant les paramètres de retour de l'opération.

- **Middleware** : C'est le logiciel qui est au milieu assure les dialogues entre les clients et les serveurs souvent hétérogènes.

II.4.Le fonctionnement de client/serveur :

Un système client/serveur fonctionne selon le schéma suivant :



II.5. Les différentes architectures client/serveur : [34]

II.5.1. L'architecture à 2 niveaux :

L'architecture à deux niveaux (aussi appelée architecture 2-tiers, tiers signifiant étage en anglais) caractérise les systèmes clients/serveurs dans lesquels le client demande une ressource et le serveur la lui fournit directement. Cela signifie que le serveur ne fait pas appel à une autre application afin de fournir le service.

II.5.2.L'architecture à 3 niveaux :

Dans l'architecture à 3 niveaux (appelées architecture 3-tiers), il existe un niveau intermédiaire ; c'est-à-dire que l'on a généralement une architecture partagée entre :

- Le client : le demandeur de ressources.
- Le serveur d'application (appelé aussi middleware) : le serveur chargé de fournir la ressource mais faisant appel à un autre serveur.
- Le serveur secondaire (généralement un serveur de bases données), fournissant un service au premier serveur.

II.5.3.L'architecture multi-niveaux :

Dans l'architecture à 3 niveaux chaque serveur effectue une tâche (un service) spécialisée. Ainsi un serveur peut utiliser les services d'un ou de plusieurs autres serveurs afin de fournir son propre

service. Par conséquent, l'architecture à 3 niveaux est potentiellement une architecture à N niveaux.

III. Le patron MVC : [35]

Le patron Modèle-Vue-Contrôleur (MVC) est un modèle de conception (Design Pattern) (il est indépendant du langage de programmation) logicielle très répandu et fort utile. Il est aujourd'hui fortement recommandé dans l'univers J2EE.
Le Design Pattern MVC est destiné à répondre aux besoins des applications interactives quand les utilisateurs manipulent des données volumineuses et complexes.

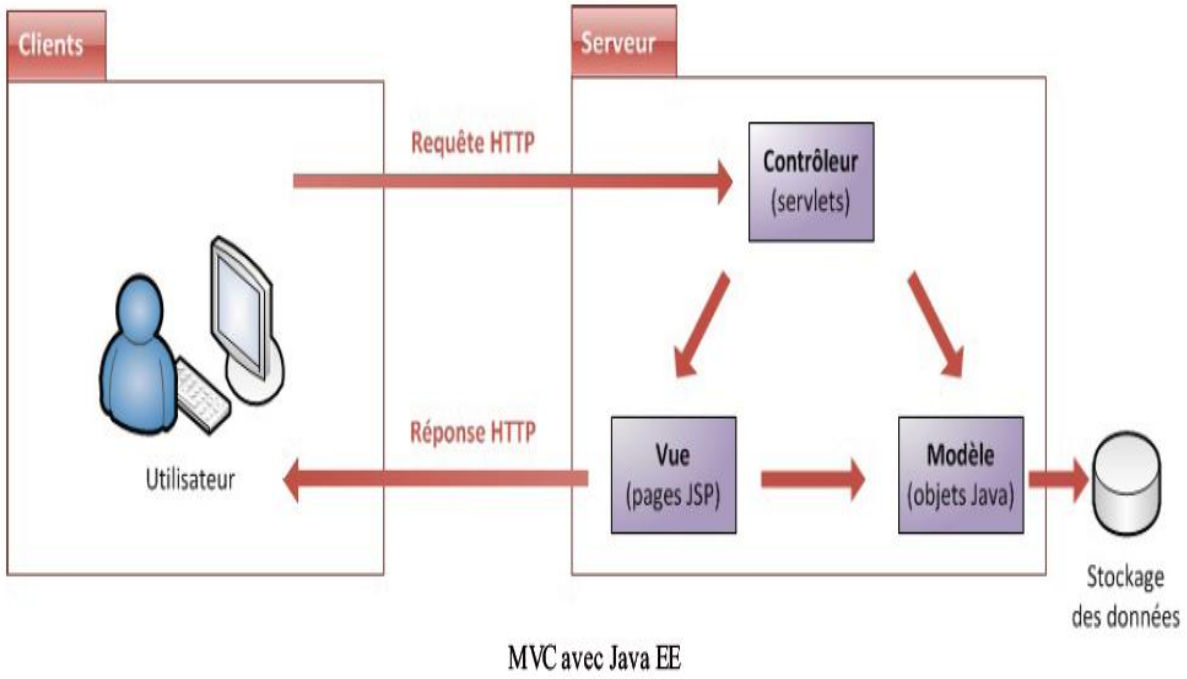


Figure IV.1: Fonctionnement du modèle MVC.

Avantages du MVC :

L'approche MVC apporte de réels avantages:

- Une conception claire et efficace grâce à la séparation des données de la vue et du contrôleur
- Un gain de temps de maintenance et d'évolution du site
- Une plus grande souplesse pour organiser le développement du site entre différents développeurs (indépendance des données, de l'affichage (webdesign) et des actions)

- **Inconvénients :**

L'inconvénient majeur du modèle MVC n'est visible que dans la réalisation de petits projets, de sites internet de faible envergure.
En effet, la séparation des différentes couches nécessite la création de plus de fichiers (3 fois plus exactement):

1. Un fichier pour le modèle
2. Un fichier pour le contrôleur
3. Un fichier pour la vue

Il n'est donc pas très intéressant de recourir à ce système dans ce cas là.

IV. Environnement et outils de développement :

1. Le Système d'exploitation Windows 7 :

Malgré sa récente sortie sur le marché des systèmes d'exploitation, cette version éditer par le géant Microsoft est très apprécié du grand public pour sa rapidité son efficacité et surtout sa maniabilité.

2. IDE Netbeans : [36]

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

Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java). Un environnement Java Development Kit JDK est requis pour les développements en Java.

NetBeans constitue par ailleurs une plate forme qui permet le développement d'applications spécifiques (bibliothèque Swing (Java)). L'IDE NetBeans s'appuie sur cette plate forme.

Dans notre cas on a utilisé la version 8.0.

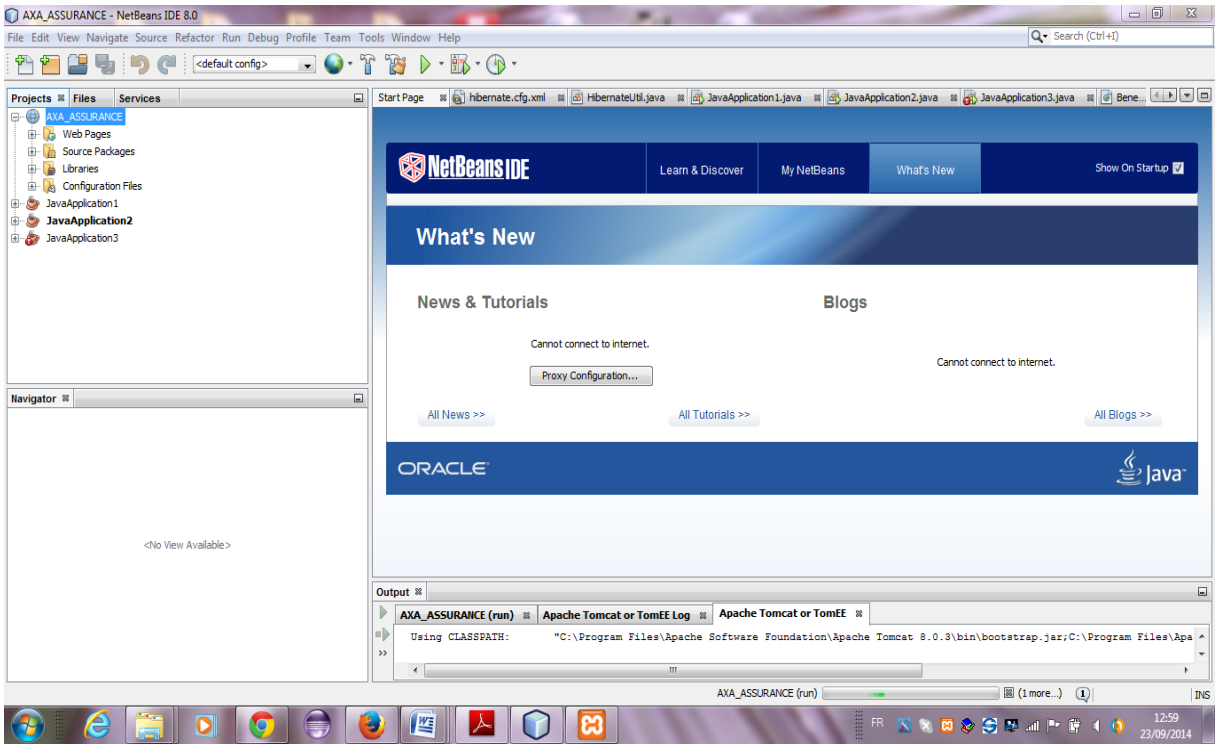


Figure IV.2. : lnterface Netbeans.

3. Plateforme J2EE: [37]

J2EE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :

- un ensemble de spécifications pour une infrastructure dans laquelle s'exécutent les composants écrits en Java : un tel environnement se nomme serveur d'applications.
- un ensemble d'API qui peut être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.

Sun propose une implémentation minimale des spécifications de J2EE : le J2EE SDK. Cette implémentation permet de développer des applications respectant les spécifications mais n'est pas prévue pour être utilisée dans un environnement de production. Ces spécifications doivent être respectées par les outils développés par des éditeurs tiers.

L'utilisation de J2EE pour développer et exécuter une application offre plusieurs avantages :

- une architecture d'applications basée sur les composants qui permet un découpage de l'application et donc une séparation des rôles lors du développement.
- la possibilité de s'interfacer avec le système d'information existant grâce à de nombreuses API : JDBC, JNDI, JMS, JCA ...
- la possibilité de choisir les outils de développement et le ou les serveurs d'applications utilisés qu'ils soient commerciaux ou libres.

J2EE permet une grande flexibilité dans le choix de l'architecture de l'application en combinant les différents composants. Ce choix dépend des besoins auxquels doit répondre l'application mais aussi des compétences dans les différentes API de J2EE. L'architecture d'une application se découpe idéalement en au moins trois tiers :

- la partie cliente : c'est la partie qui permet le dialogue avec l'utilisateur. Elle peut être composée d'une application standalone, d'une application web ou d'applets
- la partie métier : c'est la partie qui encapsule les traitements (dans des EJB ou des JavaBeans)
- la partie donnée : c'est la partie qui stocke les données.

4. Framework :

4.1. Le Framework JSF: [38]

Java server faces est un Framework de développement d'applications web en java permettant de respecter le model d'architecture MVC et basé sur des composantes coté présentation.

- Architecture MVC pour séparer l'interface utilisateur, la couche de persistance et les processus métiers, utilisant la notion d'événement, conversion des données, validation des données (par exemple des champs de formulaires requis).

Automatisation de l'affichage des messages d'erreur en cas de problèmes de conversion ou de validation.

Le premier objectif de JSF, est de procurer un environnement de développement permettant de construire une interface de type web, sans devoir toucher au code HTML et JavaScript. Ceci est réalisé par la mise en place d'un mapping entre l'HTML et les objets concernés. JSF est donc basé sur la notion de composants, comparable à celle de Swing, ou l'état de ces composants est sauvegardé puis restauré au retour de la requête.

- **faces-config.xml :**

Le faces-config.xml, est le fichier de configuration, qui permet de gérer les beans, et les règles de navigation entre les pages. Vous pouvez apercevoir, ci-dessous un bref, exemple de fichier de configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xi="http://www.w3.org/2001/XInclude"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http:

  <navigation-rule>
 <from-view-id>/*</from-view-id>
 <navigation-case>
  <from-action>#{administrateurBean.aller_client()}</from-action>
  <from-outcome>client</from-outcome>
  <to-view-id>/Client.xhtml</to-view-id>
  <redirect/>
 </navigation-case>
</navigation-rule>

   <navigation-rule>
 <from-view-id>/*</from-view-id>
 <navigation-case>
  <from-action>#{administrateurBean.aller_expert()}</from-action>
  <from-outcome>expert</from-outcome>
  <to-view-id>/Expert.xhtml</to-view-id>
  <redirect/>
 </navigation-case>
</navigation-rule>
```

Figure IV.3 : Fichier de configuration face-config.xml

- **web.xml :**

Le web.xml qui permet de définir l'emplacement des pages JSP, ainsi que de possible contraintes de sécurité concernant l'accès à des pages. Il établit également l'endroit ou se situe le Faces Servlet. Faces Servlet, est la classe, qui est le moteur de chaque application JSF. Chaque application JSF à sa propre Faces Servlet, qui gère toutes les informations relatives à la requête courante.

Voici un exemple de fichier web.xml, qui est normalement créer à la génération du projet sous Ntebeans.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Figure IV.4. : Fichier de configuration web-xml

- **Prime faces :**

Prime faces c'est l'un des composants de JSF c'est une bibliothèque open source de composantes JSF il est basé coté serveur sur l'API standard de ce dernier.

Elle permet de mettre à disposition des interfaces clients riches dans les applications web JEE.

4.2.Hibernate : [39]

Hibernate est un logiciel écrit sous la responsabilité de Gavin King, qui fait entre autre de l'équipe de développement de JBoss.

Hibernate est un Framework en open source, gérant la persistance des objets (qui peuvent être défini par les propriétés, les méthodes ou les évènements qu'il est susceptible de déclencher) dans une base de données relationnelle. La persistance des objets représente la possibilité d'enregistrement de l'état d'un objet, par exemple dans une base de données, afin de pouvoir le recréer plus tard. Quant à la base de données relationnelle, elle contient de nombreuses tables et l'information est organisée par différentes relations entre tables. Pour information, on appelle SGBDR un logiciel mettant en œuvre une telle base de données.

L'ensemble des données nécessaires au fonctionnement de l'application est sauvegardé dans une base de données. La manipulation des données peut se faire de différentes manières : par l'accès directement à la base en écrivant les requêtes SQL adéquates, utiliser un outil d'ORM (Object Relationnal Mapping) permettant de manipuler facilement les données et d'assurer leur persistance : c'est le cas d'Hibernate.

Pourquoi ajouter une couche entre l'application et la base de données ? L'objectif est de réduire le temps de développement de l'application en éliminant une grande partie du code SQL à écrire pour interagir avec la base de données et en encapsulant le code SQL résiduel. Les développeurs manipulent les classes dont les données doivent être persistantes comme des classes Java normales. Seule une initialisation correcte d'Hibernate doit être effectuée, et quelques règles respectées lors de l'écriture et de la manipulation des classes persistantes.

Hibernate se place à un autre niveau que le Framework Struts qui lui gère l'interface homme-machine et se base sur l'architecture MVC. Hibernate est agnostique en terme d'architecture, il peut donc être utilisé aussi bien dans un développement client lourd que dans un environnement Web léger de type Tomcat (Apache) ou dans un environnement J2EE complet (Weblogic, Websphere, JBoss).

Non seulement, Hibernate s'occupe du transfert des classes Java dans les tables de la base de données (et des types de données Java dans les types de données SQL), mais il permet de faire des requêtes sur les données et propose des moyens de les récupérer. Il peut donc réduire de manière significative le temps de développement qui aurait été dépensé autrement dans une manipulation manuelle des données via SQL et JDBC. Le but d'Hibernate est de libérer le développeur de 95% des tâches de programmation liées à la persistance des données communes. Hibernate n'est probablement pas la meilleurs solution pour les applications centrées sur les données qui n'utilisent que les procédures stockées pour implémenter la logique métier dans la base de données, il est plus utile dans les modèles métier orientés objets dont la logique métier est implémentée dans la couche Java dite intermédiaire. Cependant, Hibernate aide à supprimer ou à encapsuler le code SQL spécifique à la base de données et aide sur la tâche commune qu'est la transformation des données d'une représentation tabulaire à une représentation sous forme de graphe d'objets.

4.2.1. Architecture d'hibernate :

Voici une vue de l'architecture complète d'Hibernate.

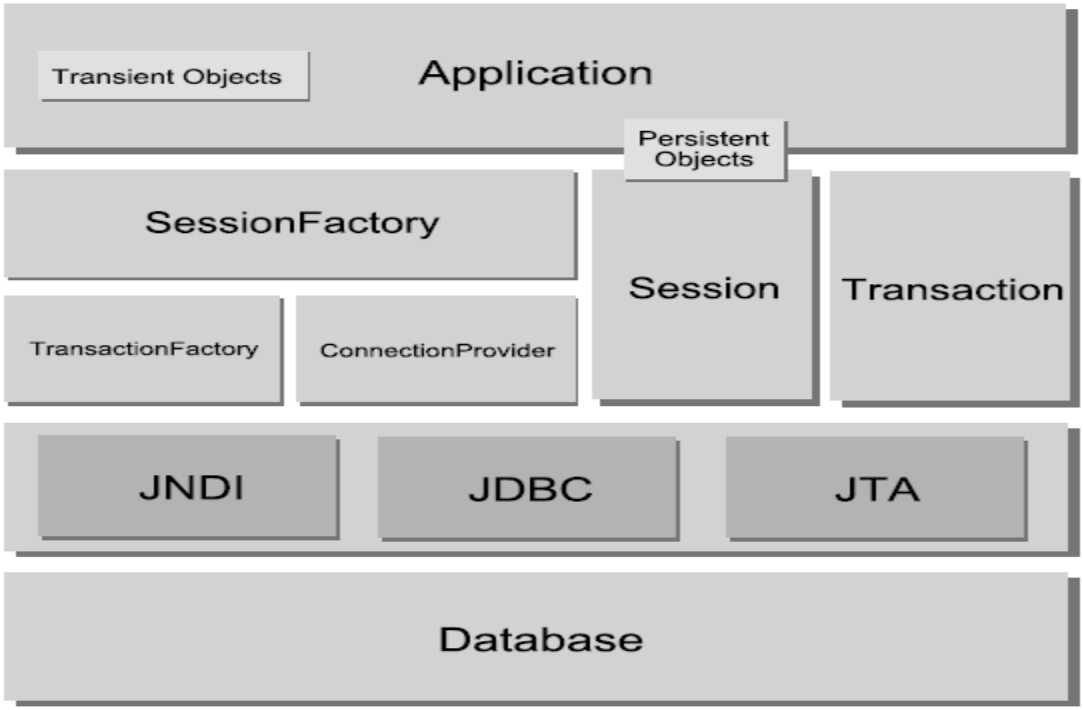


Figure IV.5. : Architecture complète d'Hibernate.

- ✓ L'architecture complète fait abstraction de l'application des API JDBC/JTA sous-jacentes et laisse Hibernate s'occuper des détails.

4.2.2. Librairies Hibernate(Librairies .jar) :

L'ajout du support de Hibernate revient à rajouter un ensemble de Jar du répertoire lib de Hibernate.

- Pour notre cas voici quelques jar:
- Com-mysql.jdbc_5.1.5.jar
- Aftwork-1.0.10.jar
- Commons-fileupload-1.2.1.jar
- Hibernate-c3po-4.3.4.Final.jar
- Jboss-logging-3.1.3.GA.jar
- Jsf-api-2.1.7.Jar
- Jstl.jar
- Ogdbc14.jar
- Primefaces-3.4.2.jar
- Servelet-api.jar

4.2.3. Fichier de configuration :

- ✓ **Fichier hibernate.cfg.xml :**

Hibernate propose des classes qui héritent de la classe Dialect pour chaque base de données supportée. C'est le nom de la classe correspondant à la base de données utilisée qui doit être obligatoirement fourni à la propriété hibernate.dialect.

Les propriétés sont alors définies par un tag <property>. Le nom de la propriété est fourni grâce à l'attribut « name » et sa valeur est fournie dans le corps du tag.

Il est possible de fournir les propriétés de configuration « Hibernate » dans un fichier hibernate.properties.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration D
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prope
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</pr
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/axa?ze
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.show_sql"/>
    <property name="hibernate.current_session_context_class">thread</property>
    <mapping resource="model/DeduireConstat.hbm.xml"/>
    <mapping resource="model/Administrateur.hbm.xml"/>
    <mapping resource="model/Contrat.hbm.xml"/>
    <mapping resource="model/Expert.hbm.xml"/>
    <mapping resource="model/Agent.hbm.xml"/>
    <mapping resource="model/Paiement.hbm.xml"/>
    <mapping resource="model/Beneficiaire.hbm.xml"/>
    <mapping resource="model/Sinistre.hbm.xml"/>
    <mapping resource="model/Client.hbm.xml"/>
    <mapping resource="model/Organisme.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Figure IV.6: Fichier hebirnate.cfg.xml

4.2.4. Fichiers xml de mapping :

Ces fichiers sont des éléments majeurs puisqu'ils vont permettre à Hibernate de faire le pont entre les classes de persistance et les sources de données.

Dans cette partie, on va présenter la du fichier de mapping relatif à la table «Clients »nommée

Client .hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<!-- Generated 19 sept. 2014 23:29:52 by Hibernate Tools 3.6.0 -->
<hibernate-mapping>
    <class name="model.Client" table="client" catalog="axa">
        <id name="idClient" type="java.lang.Integer">
            <column name="id_client" />
            <generator class="identity" />
        </id>
        <many-to-one name="agent" class="model.Agent" fetch="select">
            <column name="id_agent" not-null="true" />
        </many-to-one>
        <property name="nom" type="string">
            <column name="nom" length="30" not-null="true" />
        </property>
        <property name="prenom" type="string">
            <column name="prenom" length="30" not-null="true" />
        </property>
        <property name="adresse" type="string">
            <column name="adresse" length="70" not-null="true" />
        </property>
        <property name="telephone" type="string">
            <column name="telephone" length="15" not-null="true" />
        </property>
```

Figure IV.7: Exemple d'un fichier de mapping(classe client)

4.2.5. Classes de données :

Une classe de données est un Javabean qui va encapsuler les propriétés de la table dans ses champs private avec des getters et setters et qui a un constructeur par défaut.

```
package dao;

import java.util.List;
import model.Client;
import model.Contrat;
import model.Paiement;
import model.Sinistre;

/**
 *
 * @author boussad
 */
    public interface ClientService
    {
    public  boolean add(Client c,Integer idagent);
    public  boolean delete(Integer id);
    public  boolean modifier(Client c);
    public  Client edit(Integer id);
    public  List<Client> listeclient(String idagent);
    public  boolean enregistrer_contrat(Client client,Contrat contrat,Integer id
    public  List<Contrat> getlistecontrats(Integer idclient,String typecontrat);
    public Contrat getcontrat(String ncontrat);
    public boolean enregister_sinistre(Contrat contrat,Sinistre sinistre);
    public boolean enregister_paiement(Contrat contrat,Paiement paiement);
```

Figure IV.8: Exemple d'une classe (classe client)

4.2.6. La classe HibernateUtil :

La classe Hibernate nommée SessionFactory permet d'établir la connexion avec la source de données à partir du fichier de configuration « hibernate.cfg.xml ».On remarque que la classe

SessionFactory serait instanciée autant de fois qu'il y a de threads, il est donc plus adapté de rendre une même instance de SessionFactory accessible par les threads. Cette classe possède une méthode appelée currentSession() qui retourne la session hibernate en cours si elle existe sinon elle se charge d'ouvrir une nouvelle session.

```
package dao;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateUtil {
        public static final SessionFactory sessionFactory;

        static {
                try {
                        // Création de la SessionFactory é partir de hibernate.c
                        sessionFactory = new Configuration().configure().buildSe
                } catch (Throwable ex) {
                        // Make sure you log the exception, as it might be swall
                        System.err.println("Initial SessionFactory creation fail
                        throw new ExceptionInInitializerError(ex);
                }
        }

        public static final ThreadLocal session = new ThreadLocal();

        public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
```

Figure IV.9. : Le fichier HibernateUtil

5. Serveur Apache Tomcat : [40]

Est un conteneur web libre de servlets et JSP Java EE. Issu du projet Jakarta, c'est un des nombreux projets de l'Apache Software Foundation. Il implémente les spécifications des servlets et des JSP du Java Community Process, est paramétrable par des fichiers XML et de propriétés, et inclut des outils pour la configuration et la gestion. Il comporte également un serveur HTTP.

Apache Tomcat est un serveur web qui est une implémentation open source de logiciel des technologies Java Servlet et JavaServer Pages. Apache Tomcat est développé dans un environnement ouvert et participatif et publié sous la version de la licence Apache 2.

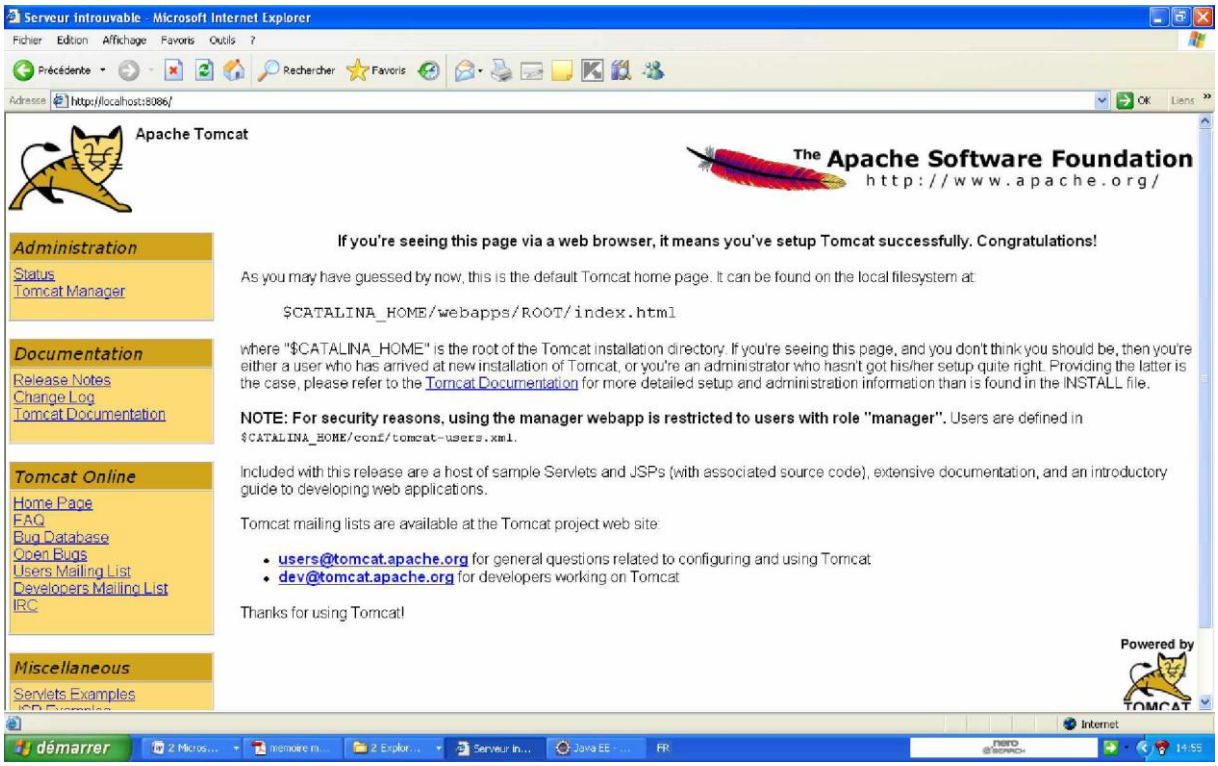


Figure IV.10. : Interface d'Apache Tomcat

V. Les langages de programmation :

V.1.Langage cote serveur :

1) Java : [41]

Java est à la fois un langage de programmation informatique orienté objet et un environnement d'exécution informatique portable créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le soutien de Bill Joy (cofondateur de Sun Microsystems En 1982), présente officiellement le 23 ai 1995 au Sun World.

Java est à la fois un langage de programmation et un environnement d'exécution. Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels qu'Unix, Microsoft Windows , Mac OS ou Linux avec peu ou pas de modifications... C'est la plate-forme qui garantit la portabilité des applications développées en Java.

Le langage reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épurée des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que l'héritage multiple remplacé par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombres entiers, nombres à virgule flottante, etc.).

Java permet de développer des applications autonomes mais aussi, et surtout, des applications client-serveur. Côté client, les applets sont à l'origine de la notoriété du langage. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets, le pendant serveur des applets, et plus récemment les JSP (JavaServer Pages) qui peuvent se substituer à PHP, ASP et ASP.NET.

Les applications Java peuvent être exécutées sur tous les systèmes d'exploitation pour lesquels a été développée une plate-forme Java, dont le nom technique est JRE (Java Runtime Environment - Environnement d'exécution Java). Cette dernière est constituée d'une JVM (Java Virtual Machine - Machine Virtuelle Java), le programme qui interprète le code Java et le convertit en code natif. Mais le JRE est surtout constitué d'une bibliothèque standard à partir de laquelle doivent être développés tous les programmes en Java. C'est la garantie de portabilité qui a fait la réussite de Java dans les architectures client-serveur en facilitant la migration entre serveurs, très difficile pour les gros systèmes**.**

2) Langage SQL: [42]

Le SQL (Structured Query Language) est un langage permettant de communiquer avec une base de données. Ce langage informatique est notamment très utilisé par les développeurs web pour communiquer avec les données d'un site web. SQL.sh recense des cours de SQL et des explications sur les principales commandes pour lire, insérer, modifier et supprimer des données dans une base.

3) Langage HQL : [43]

C'est le langage de requêtes du framework Hibernate. Il permet d'effectuer des requêtes sur une base de données de manière orientée objet. Ces requêtes seront ensuite transformées par Hibernate en requêtes SQL natives pour être exécutées sur la base de données. Les fonctionnalités du HQL s'approchent de celles du SQL, mais sont plus orientées objet, on utilise des propriétés des classes de notre modèle objet.

V.2.Le langage coté client :

1).Le langage XHTML : [44]

XHTML (Extensible HyperText Markup Language) est un langage de balisage servant à écrire des pages pour le World Wide Web. Conçu à l'origine comme le successeur de HTML, XHTML se fonde sur la syntaxe définie par XML, plus récente, mais plus simple que celle définie par SGML sur laquelle repose HTML. Il s'agissait en effet à l'époque de tirer parti des bénéfices techniques attendus de la simplification offerte par XML.

2).HTML:

L'*Hypertext Markup Language*, généralement abrégé HTML, est le format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et de

mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des programmes informatiques. Il permet de créer des documents interopérables avec des équipements très variés de manière conforme aux exigences de l'accessibilité du web. Il est souvent utilisé conjointement avec des langages de programmation (JavaScript) et des formats de présentation (feuilles de style en cascade). HTML est initialement dérivé du*Standard Generalized Markup Language* (SGML).

3). XML : [45]

L'*Extensible Markup Language* (XML, « langage de balisage extensible » en français) est un langage informatique de balisage générique qui dérive du SGML. Cette syntaxe est dite « extensible » car elle permet de définir différents espaces de noms, c'est-à-dire des langages avec chacun leur vocabulaire et leur grammaire, comme XHTML, XSLT, RSS, SVG… Elle est reconnaissable par son usage des chevrons (< >) encadrant les balises. L'objectif initial est de faciliter l'échange automatisé de contenus complexes (arbres, texte riche…) entre systèmes d'informations hétérogènes (interopérabilité). Avec ses outils et langages associés, une application XML respecte généralement certains principes :

- la structure d'un document XML est définie et validable par un schéma,
- un document XML est entièrement transformable dans un autre document XML.

4).Java script : [46]

JavaScript est un langage de programmation de scripts principalement utilisé dans les pages web interactives mais aussi côté serveur. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés. En outre, les fonctions sont des objets de première classe.

Le langage a été créé en 1995 par Brendan Eich (Brendan Eich étant membre du conseil d'administration de la fondation Mozilla à cette époque) pour le compte de Netscape Communications Corporation. Le langage, actuellement à la version 1.8.2, est une implémentation de la 3^{e} version de la norme ECMA-262 qui intègre également des éléments inspirés du langage Python. La version 1.8.5 du langage est prévue pour intégrer la 5^{e} version du standard ECMA.

5).Le SGBD MSQL : [47]

MySQL est un système de gestion de base de données (SGBD).Selon le type d'application, sa licence est libre ou propriétaire.

Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle ou Microsoft SQL Server.

6). PHP My Admin : [48]

6.1.PhpMyAdmin :

Le PhpMyAdmin est une application web de gestion pour les systèmes de gestion de base de données MYSQL réalisées en PHP en passant par un navigateur web.

Le PhpMyAdmin autorise un ensemble de fonctions sur une base de données, parmi elles on cite :

• La création et la suppression d'une base de données.
• La création, la modification et la suppression d'une table de la base de données.
• L'ajout et la suppression des champs.
• L'exécution des commandes SQL.
• La création des index.
• Le chargement de fichiers textes dans des tables.
• La gestion des privilèges des utilisateurs

- **Utilisation de PhpMyAdmin :**

Pour accéder à PhpMyAdmin, il faut d'abord vérifier qu'Easyphp.exe est lancer et que le serveur fonctionne, après on peut accéder à partir de « l'administration ». Pour ouvrir l'administration il suffit de faire un clic droit sur l'icône et sélectionner ''administration''. Une page Web apparaît, au milieu de celle-ci il y a un bouton PhpMyAdmin avec un simple clique là-dessus, la page d'accueil de PhpMyAdmin s'affiche dans la fenêtre du navigateur, accompagner d'un champ de sélection de base de données présente sur l'hôte MySQL par défaut, comme l'illustre la figure suivante :

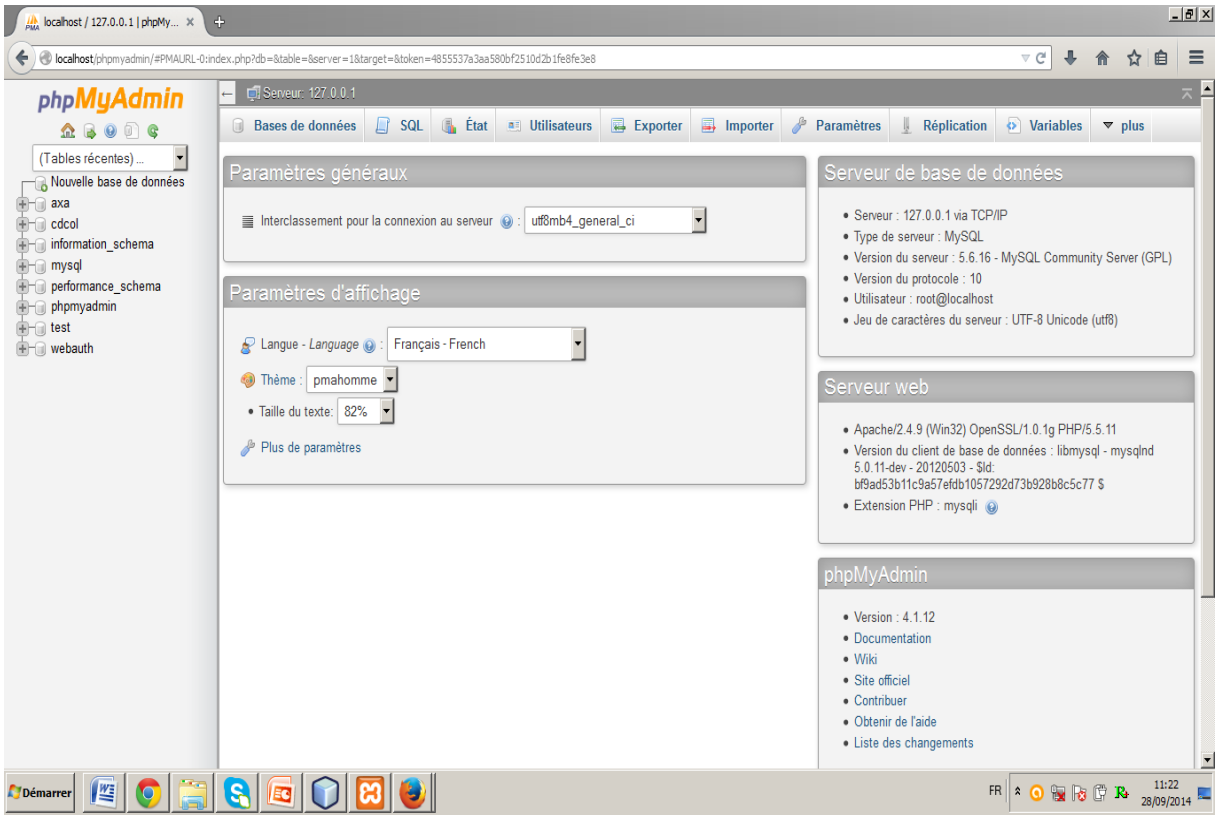


Figure IV.11 : Administration de MySQL à partir de PhpMyAdmin

VI. Implémentation de la base de données :

- **Table personne :**

Nom du champ	Description du champ	Type de donnée	Clé (s)
Id_personne	Identifiant de la personne	Int(11)	PRIMAIRE
Nom_personne	Nom de la personne	Varchar(30)	
Prenom _ personne	Prénom de la personne	Varchar(30)	

- **Table agent :**

Nom du champ	Description du champ	Type de donnée	Clé (s)
Id_agent	Identifiant de l'agent	Int(11)	PRIMAIRE
Login-agent	login de l'agent	Varchar(8)	
Mdp_agent	Mot de passe	Varchar(8)	
Nom_agent	Nom de l'agent	Varchar(30)	
Prenom _ agent	Prénom de l'agent	Varchar(30)	

- **Table administrateur :**

Nom du champ	Description du champ	Type de donnée	Clé(s)
Id_admin	Identifiant de l'administrateur	Int (11)	Clé PRIMAIRE
Login_admin	Identifiant de l'administrateur	Varchar(10)	
Mdp_admin	Mot de passe	Varchar(10)	

- **Table client :**

Nom du Champ	Description du champ	Type de donnée	Clé(s)
Id_client	Identifiant de client	Int(11)	Primaire
Nom_client	Nom de client	Varchar(30)	
Prénom_client	Prénom de client	Varchar(30)	
Téléphone_client	Numéro téléphone de client	Varchar(15)	
Prof-client	Profession du client	Varchar(70)	
Adresse_client	Adresse de client	Varchar(70)	
Id_agent	Identifiant de l'agent	Int(11)	Etrangére

- **Table contrat :**

Nom du champ	Description du champ	Type de donnée	Clé (s)
Id_contrat	Identifiant de contrat	Bigint(20)	Primaire
Num_contrat	Numéro de contrat	Varchar(11)	Primaire
Montant_paye	Montant payé	Double	
Type_contrat	Type de contrat	Varchar(30)	
Desig_contrat	Désignation de contrat	Varchar(40)	
Date_contrat	Date de contrat	Date	
Etat	L'état de contrat	Varchar(36)	
Date_décés	Date décès	Date	
Taux_ rembours	Taux de remboursement	Varchar(12)	
Date_hosp	Date de l'hospitalisation	Date	
Date_reactivation	Date de réactivation du contrat	Date	
Dat_Validité	Date de validité de contrat	Date	
Type_hosp	Type d'hospitalisation	Varchar(45)	Etrangère
Id_organisme	Identifiant de l'organisme	Int(11)	Etrangère
Id_Client	Identifiant de client	Int(11)	

- **Table sinistre :**

Nom du champ	Description du champ	Type de donnée	Clé(s)
Id_sinistre	Identifiant de sinistre	Int(11)	Primaire
Design_sinistre	Designation de sinistre	Varchar(70)	
Type_sinistre	Type de sinistre	Varchar(25)	
Date_sinistre	Date de sinistre	Date	
Id_contrat	Identifiant de contrat	Bigint(20)	Etrangère

- **Table bénéficiaire :**

Nom du champ	Description du champ	Type de donnée	Clé(s)
Id_bénéf	Identifiant de bénéficiaire	bigint(20)	Primaire
Id_client	Identifiant de client	Int(11)	Etrangère
Nom_bénéf	Nom de bénéficiaire	Varchar(45)	
Prénom_bénéf	Prénom de bénéficiaire	Varchar(45)	
Age_bénéf	Age de bénéficiaire	Int(11)	

- **Table expert :**

Nom du champ	Description du champ	Type de donnée	Clé(s)
Id_expert	Identifiant de l'expertt	Int(11)	Primaire
Nom_expert	Nom de l'expert	Varchar(30)	
Prénom_expert	Prénom de l'expert	Varchar(40)	
Adresse_expert	Adresse de l'expert	Varchar(50)	
Specialité_expert	Spécialité de l'expert	Varchar(15)	
Télépho_expert	Numéro de téléphone de l'expert	Varchar(15)	

- **Table profile :**

Nom de champ	Description du champ	Type de donnée	Clé(s)
Id_prof	Identifiant de profile	Int(11)	Primaire
Titre_prof	Titre de profile	Varchar(25)	

VII. Présentation de quelques interfaces de l'application :

Nous allons, dans ce qui suit, présenter les principales interfaces illustrent le fonctionnement de notre application.

VII.1. Page Authentification :

Cette page est nécessaire pour l'authentification des différents acteurs qui vont utiliser cette application (Agent, administrateur), ce qui va leur permettre d'accéder à leur espace personnel et pour cela il suffit de cliquer sur le lien authentification ensuite remplir le formulaire (Login, Mot de passe) et valider la saisie :

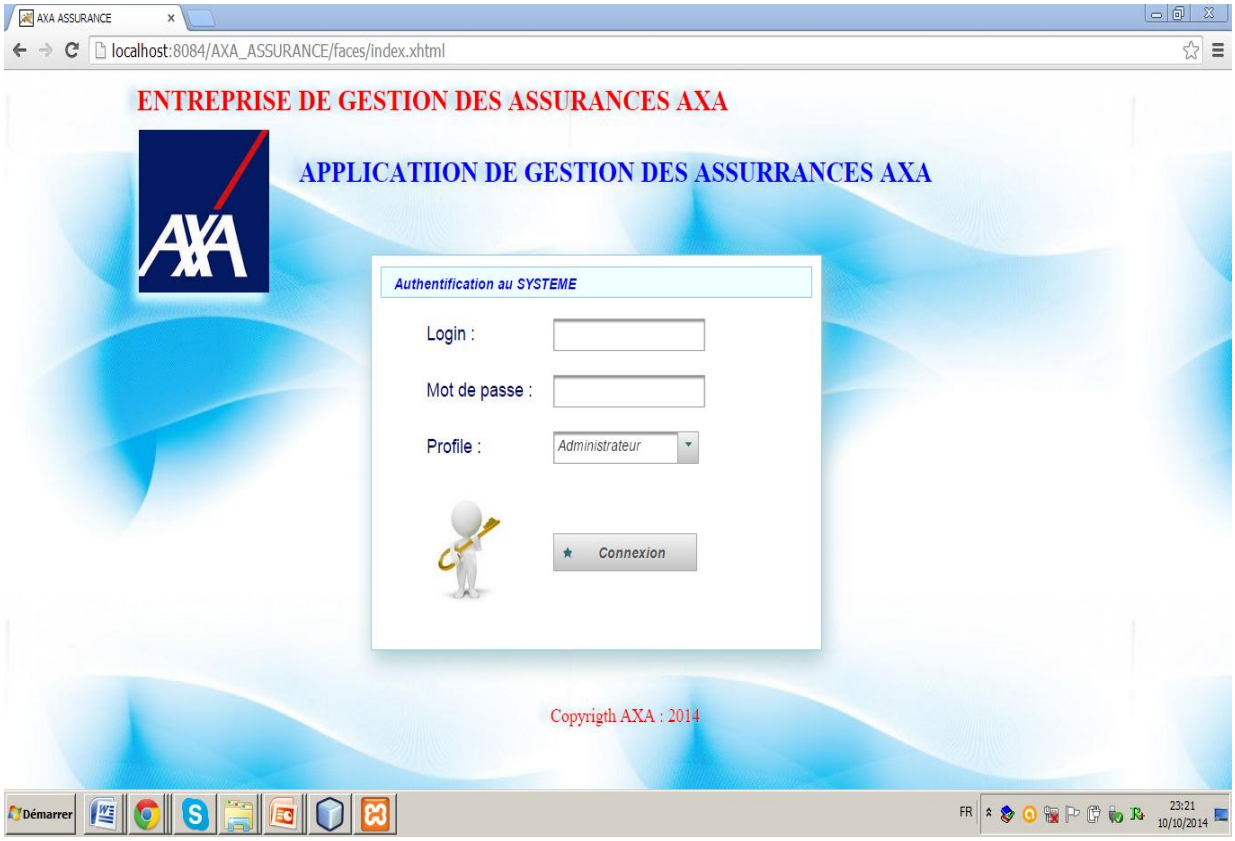


Figure IV.12 : La page Authentification

En cas d'introduction de données erronées lors de l'authentification un message d'erreur est affiché.

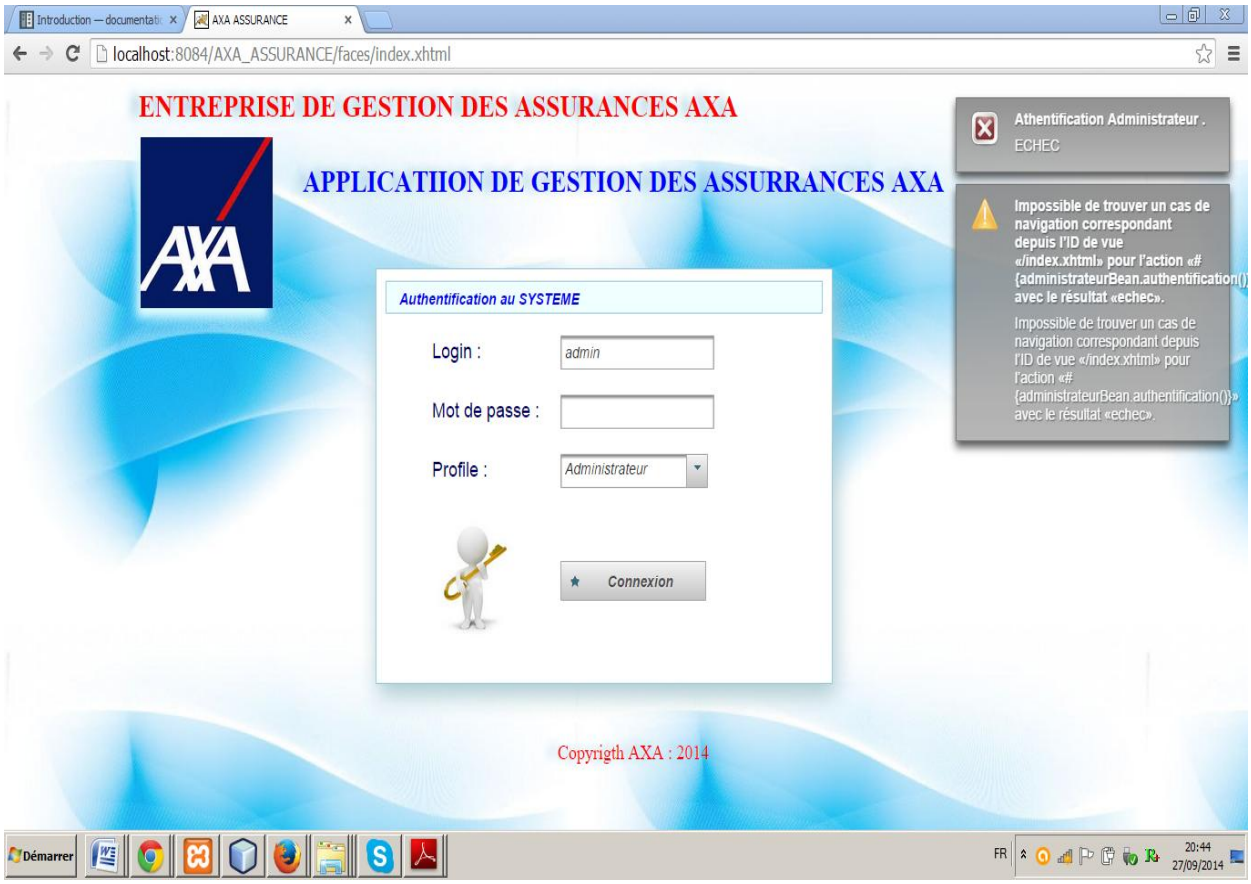


Figure IV.13 : La page d'erreur d'Authentification

VII.2.La page Espace Personnel :

Cette page peut être atteinte par tout Acteur ayant un compte, ce dernier peut l'atteindre après avoir effectué l'authentification, il pourra effectuer plusieurs tâches.

VII.2.1.Page espace agent :

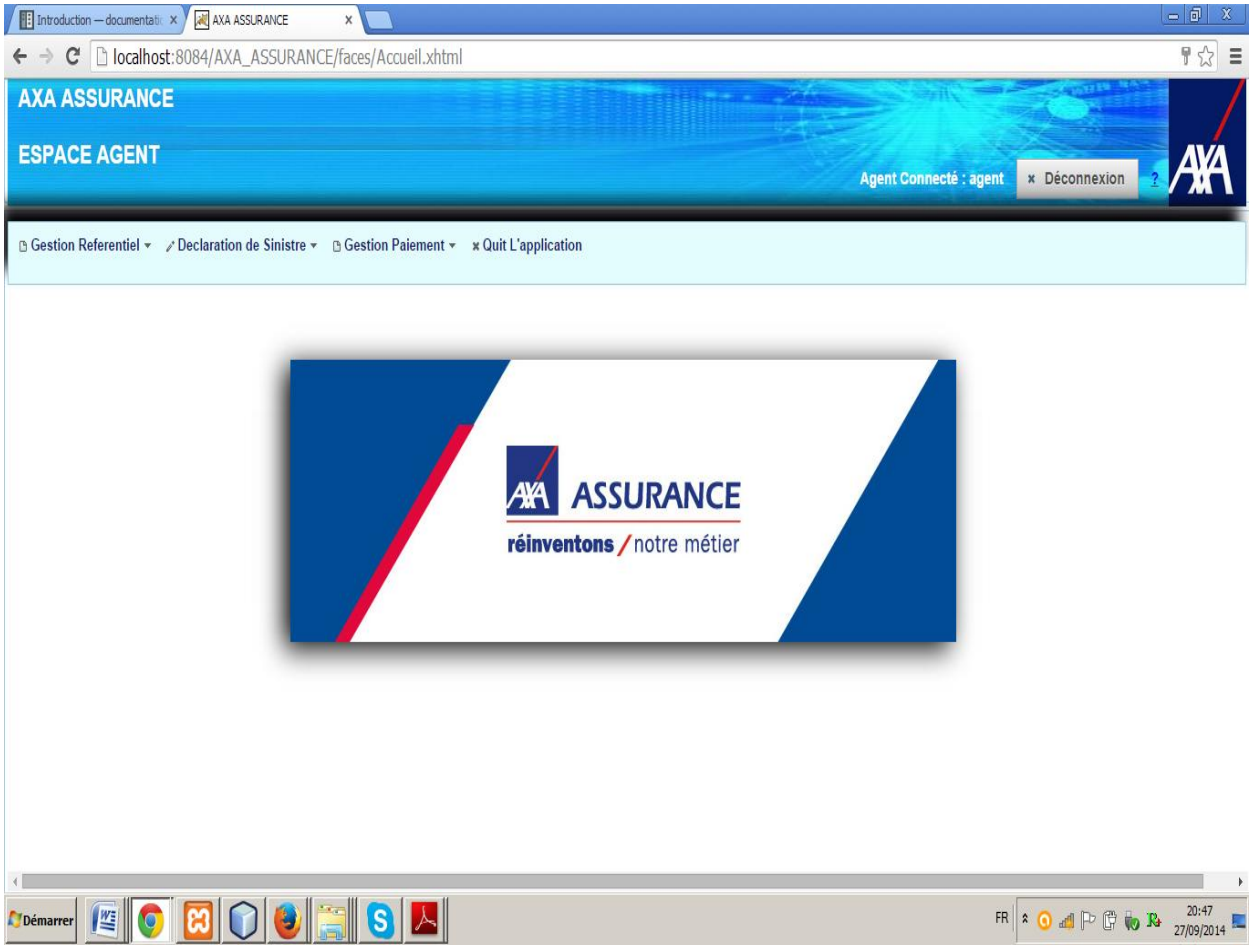


Figure IV.14 : La page Espace Agent.

VII.2.2.Page espace administrateur :

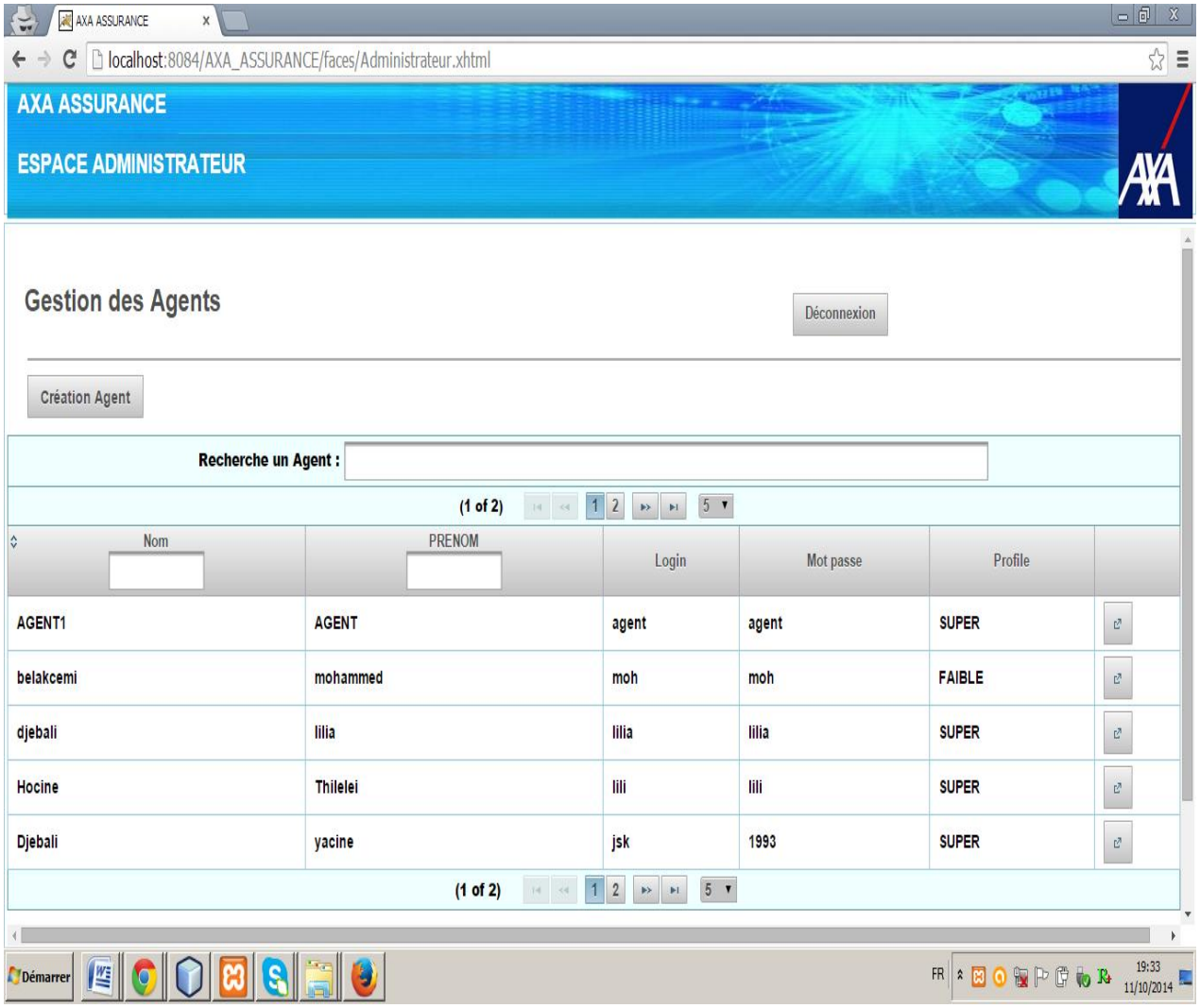


Figure IV.15 : La page Espace Administrateur.

VII.3. Page Ajouter un agent :

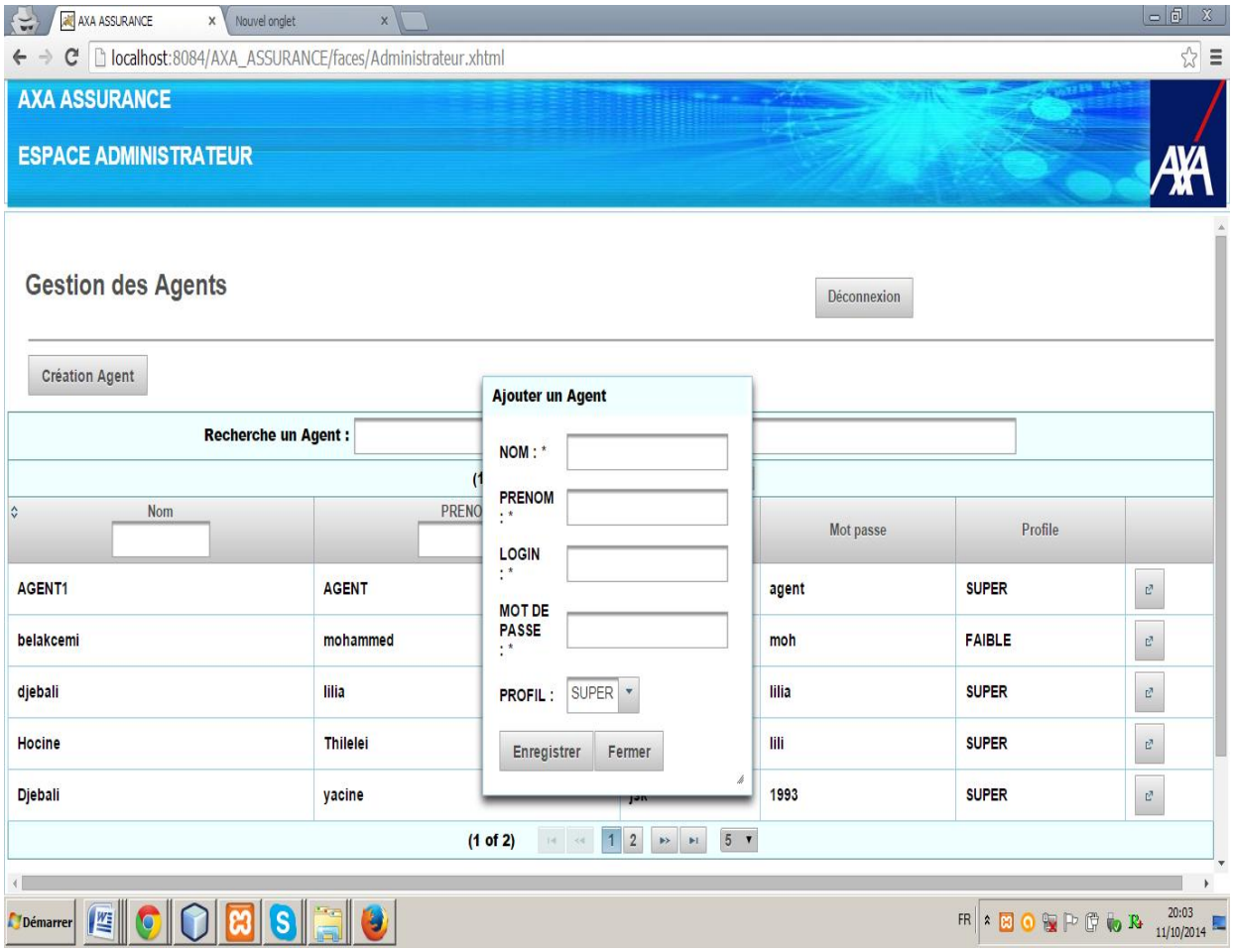


Figure IV.16 : La page Ajouter un agent.

VII.4.Page Ajouter un client :

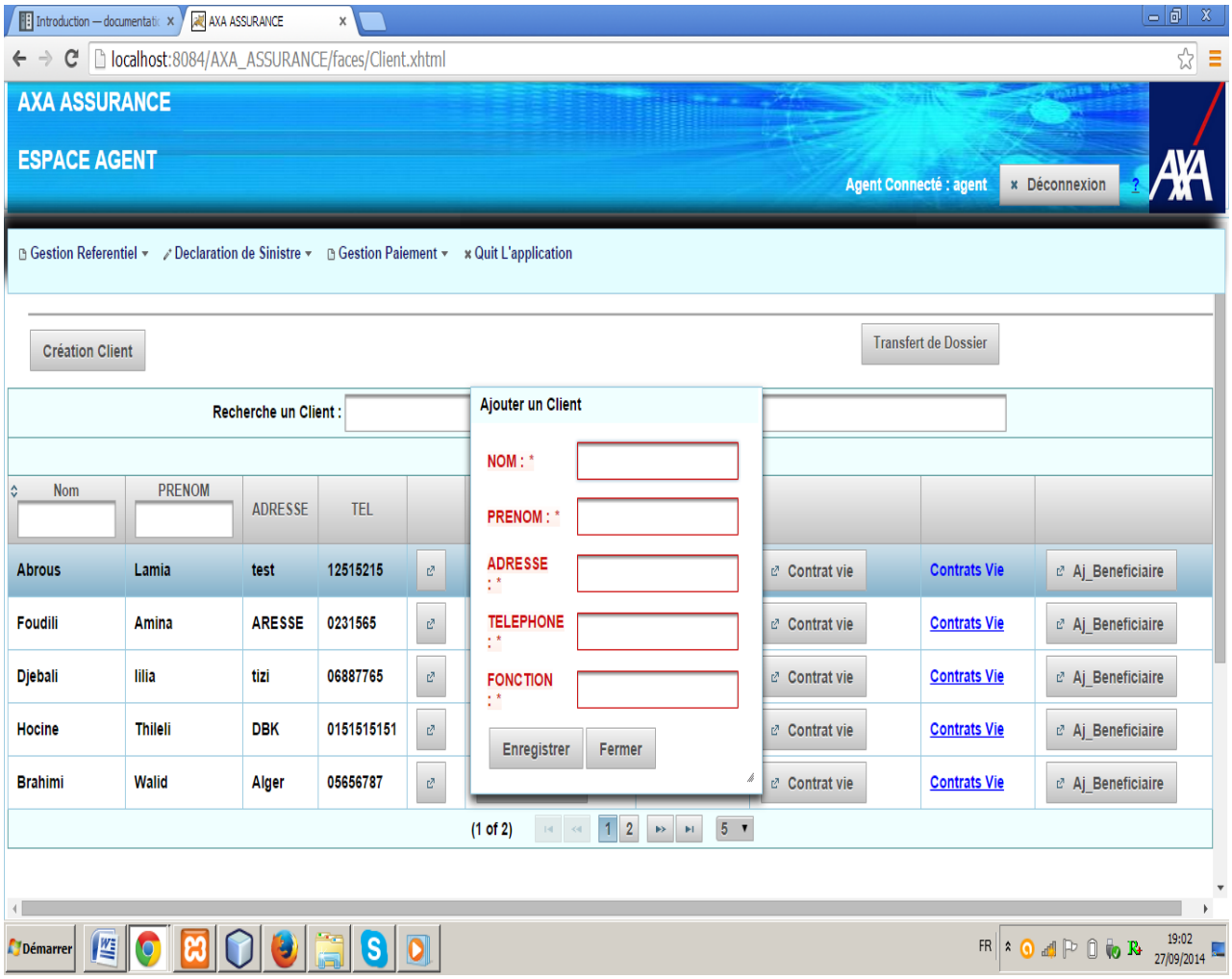


Figure IV.17 : La page Ajouter un client.

VII.5.Page Modification Client :

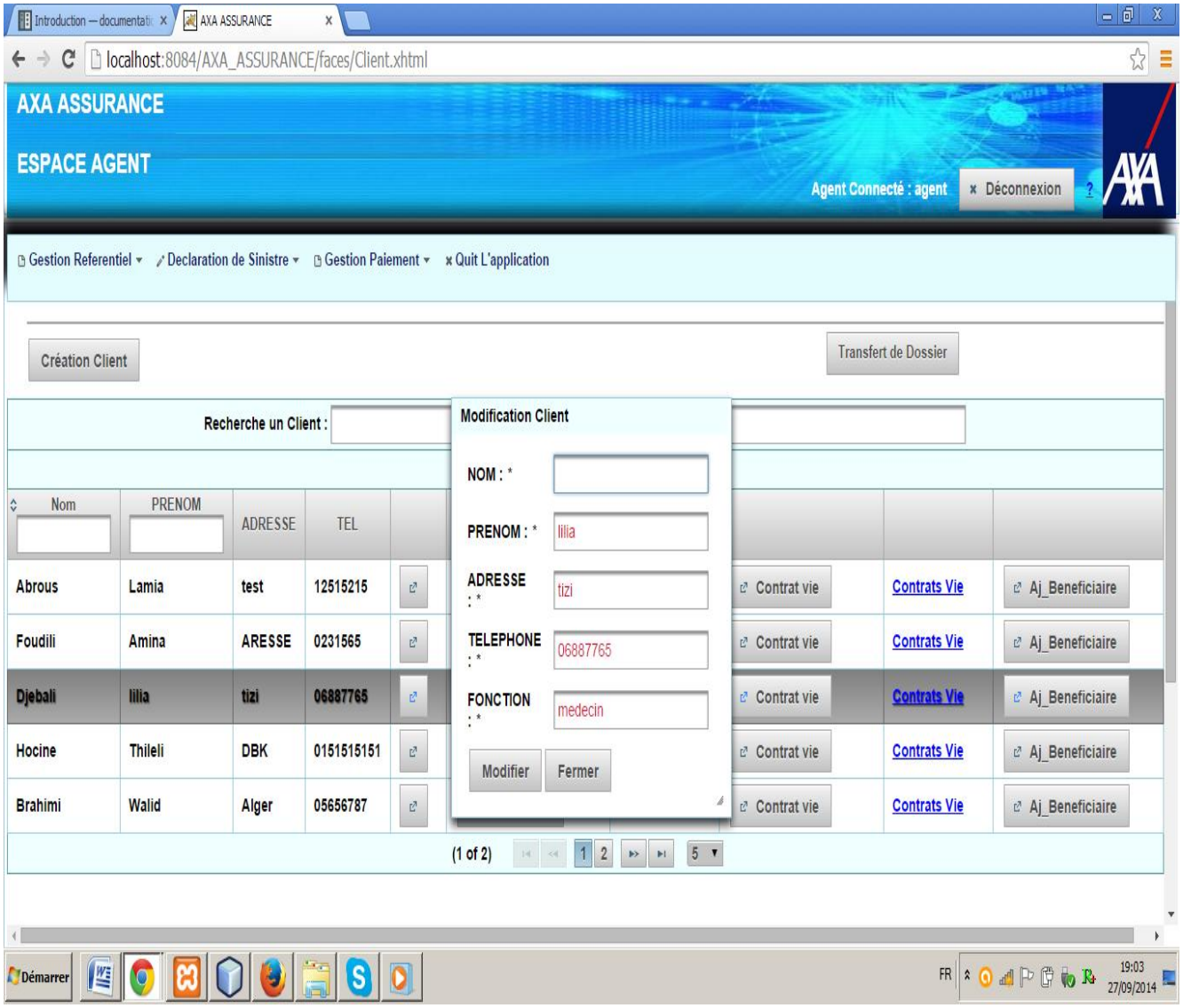


Figure IV.18 : La page Modification un client.

VII.6.Page création d'un contrat de vie :

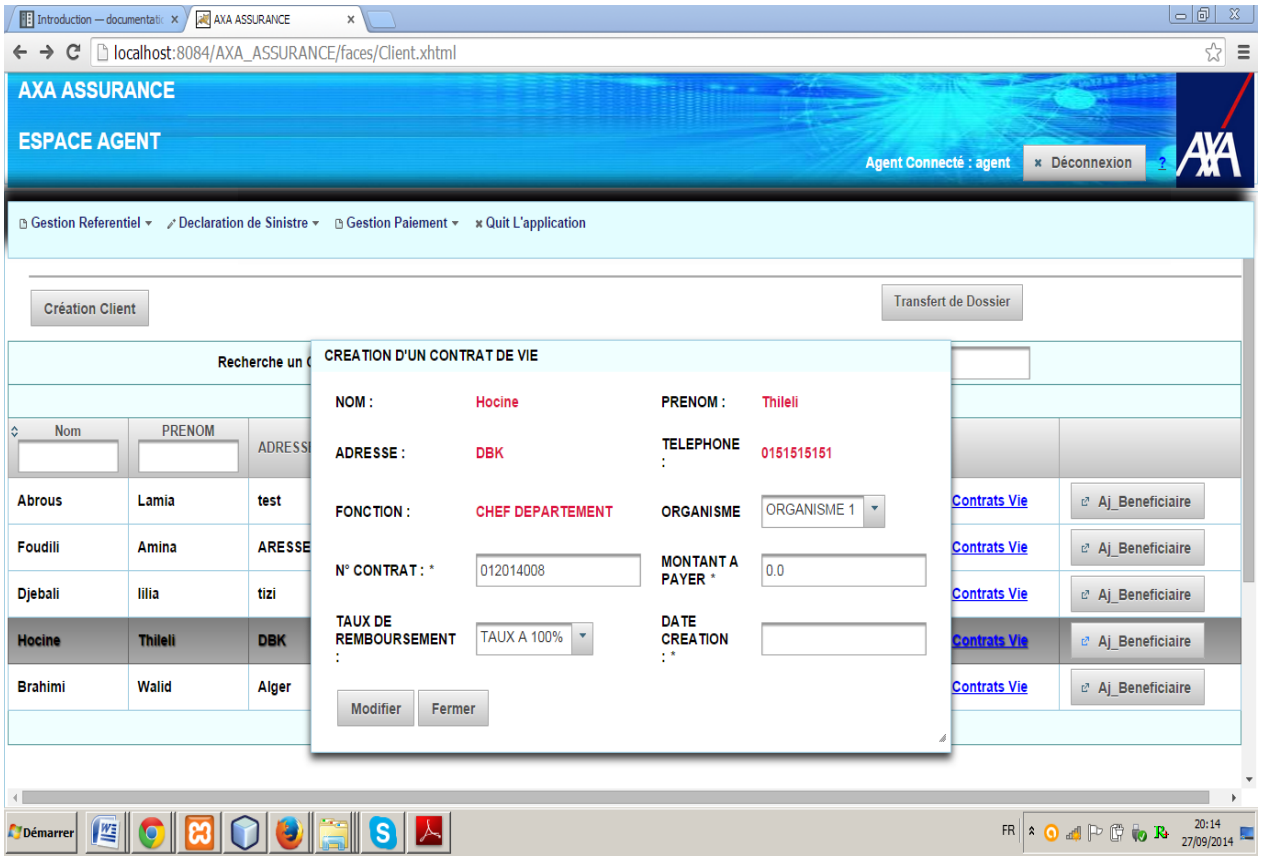


Figure IV.19 : La page Création un contrat de vie.

VIII. Conclusion :

Nous avons abordé dans ce chapitre les différents outils de développement et les langages de programmations utilisés pour le développement dans notre application ainsi que quelques fonctionnalités qu'effectue cette dernière.

Conclusion générale

Conclusion générale

L'objectif de notre projet été présentation des méthodes agiles, UML pour la conception d'une application Web pour une société d'assurance AXA.

Pour mener à terme ce mémoire, nous avons donné un aperçu général sur les méthodes agiles, ces méthodologies se caractérisent par un attachement farouche à tout planifier, « tout doit être prévisible », en tout début de projet. Voilà pourquoi on les qualifie d'approches «prédictives ». Un plan de management du projet décrit comment et quand le travail sera réalisé, les modalités de planification, d'exécution, de suivi et de clôture du projet.

Ensuite, nous avons également parlé sur les généralités des assurances particulier le groupe AXA qui a été 'objectif de notre application.

Après avoir effectué l'analyse et la conception de notre système en utilisant l'extension d'UML pour le web ainsi que la conception et l'implémentation de notre base de données en recensant les différentes informations manipulées, nous avons terminé par la réalisation qui nous a donné l'occasion d'acquérir de nouvelles connaissances et d'en approfondir d'autres sur le développement des applications Web, on citera le langage orienté objet JAVA précisément sous la plateforme J2EE, l'utilisation de framework Hibernate et ses différents outils fiables, le langage de requêtes SQL et le HTML. Ça nous a permis aussi de nous familiariser avec un certain nombre d'outils informatiques de développement, à titre d'exemple, on cite l'IDE NetBeans et MySQL.

En fin, on prévoit d'apporter d'éventuelles améliorations et nous espérons que notre travail sera de grands intérêts et un guide efficace pour les nouvelles promotions.

Références Bibliographiques

Webographie et Bibliographie

[1] http://www.entreprise-agile.com/HistoAgile.pdf

[2] http://www.journaldunet.com/developpeur/expert/56616/la-methode-agile---optimisation-de-la-relation--client---fournisseur.shtml

[3] http://www.amj-groupe.com/a141_methodes-agiles-les-principales-methodes agiles.html

[4] http://anubis.polytech.unice.fr/iut/_media/2012_2013/lp/idse/gl/methodes-partie2-xp.pdf

[5] http://extremeprogramming.free.fr/page.php?page=fondements

[6] http://prezi.com/3fw9xddkvbdg/cycle-de-vie-dun-produit-methode-agile-xp/

[7] http://ineumann.developpez.com/tutoriels/alm/agile_scrum/

[8] http://fr.wikipedia.org/wiki/Unified_Process

[9] http://fr.wikipedia.org/wiki/Dynamic_systems_development_method

[10] http://www.rad.fr/BIO/MethodeAgile.pdf

[11] http://www.access-dev.com/access-dev/la-gestion-de-projet-methodes-classiques-vs-methodes-agiles/

[12] http://fr.wikipedia.org/wiki/Crystal_clear*

[13] http://fr.wikipedia.org/wiki/M%C3%A9thode_agile

[14] Véronique Messager Rota Préface de Jean Tabaka, Gestion des projets : Vers les méthodes agiles Groupe Eyrolles ,2008

[15] http://fr.wikipedia.org/wiki/Assurance

[16] http://pro.empruntis.com/assurance/guide/les-differents-types-d-assurances

[17] http://fr.wikipedia.org/wiki/Assurance-vie

[18] http://www.ffsa.fr/sites/jcms/c_76125/fr/assurance-vie-les-contrats-dassurance-en-cas-de-vie

[19] http://assurancesplus.blogspot.com/2011/04/roles-de-lassurance.html

[20] http://fr.wikipedia.org/wiki/Compagnie_d%27assurances

[21] http://www.eldjazaircom.dz/index.php?id_rubrique=291&id_article=2819

[22] http://www.algerieassurance.com/

[23] http://www.djazairess.com/fr/lemaghreb/26390

[24]http://fr.wikipedia.org/wiki/Cat%C3%A9gorie:Compagnie_d%27assurances_fran%C3%A7aise

[25] http://www.axa.com/fr/groupe/histoire/

[26] http://www.axa.com/fr/actualites/2012/principes_assurance_responsable.aspx

[27] http://www.axa.dz/

[28] http://www.axa.dz/s-14-axa-en-algerie

[29] Mémoire licence : Mr BELABIOD Smail, Mr BELKADA Nacer, conception et réalisation d'une application web dynamique ECOTECH

[30] Pascal Roques, « UML par la pratique : Etude de cas et exercices corrigés », Edition Eyrolles.

[31] [NK01] Pascal PARE Camille ROSENTHAL-SABROUX Nasser KETTANI,

Dominique MIGNET. De Merise _a UML. Eryolles France édition, Octobre 2001.

[32] Chantal Morley Jean Hugues Bernard Leblanc, " UML pour l'analyse d'un système D'information" DUNOD

[33] http://www.definitions-webmarketing.com/Definition-Web-application

[34] George et Olivier GARDARIN, **"Le Client-serveur".** Eyrolles, 1996.

[35] Adt « Guide du designer MVC », Atelier logiciel.

[36] http://linuxfr.org/news/java-8-et-netbeans-8-sont-disponibles

[37] http://www.jmdoudoux.fr/java/dej/chap-j2ee-javaee.html

[38] http://fr.wikipedia.org/wiki/JavaServer_Faces

[39] http://fr.wikipedia.org/wiki/Hibernate

[40] http://fr.wikipedia.org/wiki/Apache_Tomcat

[41] http://ipeti.forumpro.fr/t21-definition-de-langage-java-java-script

[42] http://sql.sh/

[43]http://dico.developpez.com/html/3040-Langages-HQL-Hibernate-Query-Language.php

[44] http://fr.wikipedia.org/wiki/Extensible_HyperText_Markup_Language

[45] http://fr.wikipedia.org/wiki/Extensible_Markup_Language

[46] http://fr.wikipedia.org/wiki/JavaScript

[47] http:://www.jcc40.fr/wp-content/uploads/2012/03/mysql.pdf

[48] http://www.phpmyadmin.net/home_page/index.php

[49] Grady B James R « guide de l'utilisateur UML » Edition Eyrolles.

[50] Mémoire Ingénieur : M.A.AIT OUARABI, M.K.AIT GHERBI Conception et réalisation d'une application mobile avec interconnexion à une base de données de type multimédia à distance, Tizi Ouzou 2009-2010

[51] Jim Conallen « Concevoir des applications web avec UML », Edition Eyrolles.

ANNEXE

La notation UML

I. Définition: [49]

UML (Unified Modeling Language), que l'on peut traduire par « langage de modélisation unifié », est une notation permettant de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existant auparavant, il est devenu désormais la référence en terme de modélisation objet, à un tel point que sa connaissance est souvent nécessaire pour obtenir un poste de développeur objet.

UML n'est pas une méthode, dans la mesure où elle ne représente aucune démarche. A ce titre, UML est un formalisme de modélisation objet. Le mot méthode parfois utilisé par abus de langage ne doit pas donc être entendu comme une démarche.

II. modélisation avec UML : [50]

II.1. Définition d'un modèle :

Un modèle est une simplification et/ou une abstraction de la réalité. Il doit aider à mieux comprendre, percevoir les relations et les interactions à l'intérieur du système. Il doit permettre de visualiser les conséquences de modifications apportées au système, de visualiser également les raisons du comportement du système par rapport à une situation donnée. C'est donc un guide pour construire un système stable et fiable. Le modèle doit également aider à documenter le système construit.

II.2 La modélisation UML : [50]

UML fournit un ensemble d'outils permettant de représenter des éléments du monde objet (classes, objets, ...) ainsi que les liens qui les relient. Toutefois, étant donné qu'une seule représentation est trop subjective, UML fournit un moyen astucieux permettant de représenter diverses projections d'une même représentation grâce aux **vues**. Une vue est constituée d'un ou plusieurs **diagrammes**.

On distingue deux types de vues:

- **Les vues statiques :** Représentant le système physiquement
  - diagrammes d'objets.
  - diagrammes de classes.
  - diagrammes de cas d'utilisation.
  - diagrammes de composants.
  - diagrammes de déploiement.
- **Les vues dynamiques** : montrant le fonctionnement du système
  - diagrammes de séquence.
  - diagrammes de collaboration.
  - diagrammes d'états transitions.
  - diagrammes d'activités.

III. Les éléments d'UML :

III.1 Les éléments structurels :

Les éléments de structure sont les noms des modèles UML.

a. ***La classe :*** C'est une description abstraite, condensée d'un ensemble d'objets et de l'application. Les classes sont représentées par des rectangles compartimentés. Le premier compartiment contient le nom de la classe, nom qui doit permettre de comprendre ce que la classe est, et non ce qu'elle fait ; le deuxième compartiment contient les attributs ; le troisième contient les opérations et le quatrième compartiment facultatif représente les responsabilités. Notons que les attributs et les opérations peuvent être d'une visibilité publique, protégée ou privée.

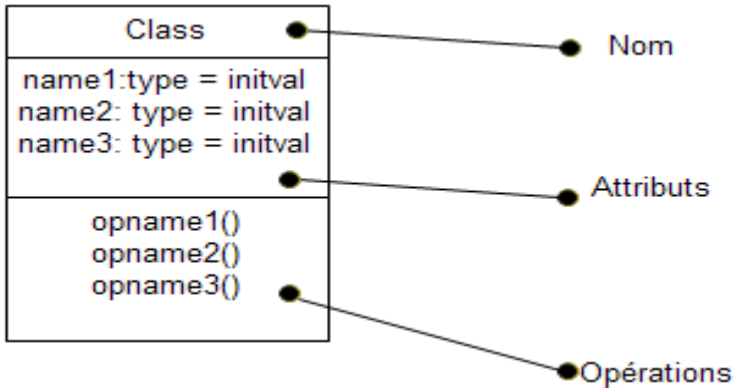


Figure A.1 : Schéma représentatif d'une classe.

b. ***Les classes-associations :*** Il est possible de représenter une association par une classe pour ajouter, par exemple, des attributs et des opérations dans l'association. Une classe de ce type, appelée **classe associative** ou **classe-association**, possède à la fois les caractéristiques d'une classe et d'une association, et peut à ce titre participer à d'autres relations dans le modèle. La notation utilise une ligne pointillée pour attacher une classe à une association. Dans la figure suivante, l'association entre les classes **A** et **B** est représenté par la classe **C**.

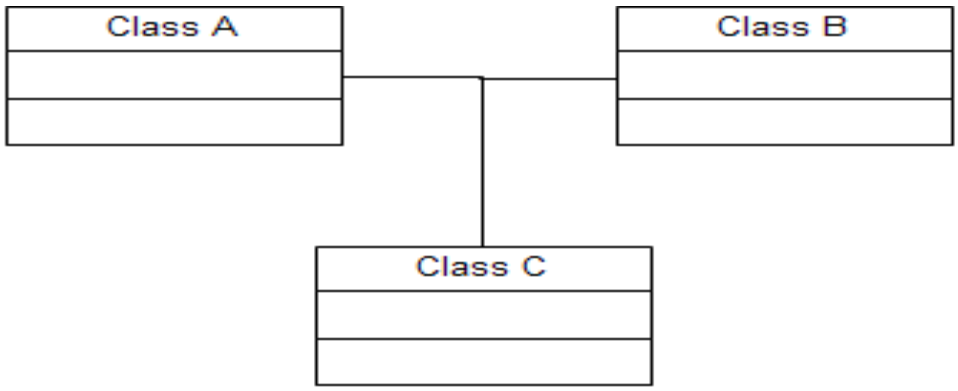


Figure A.2 : Schéma représentatif d'une classe-association.

c. ***Les interfaces :*** Une interface est une collection de spécifications d'opérations qui définissent un service d'une classe ou d'une composante. Graphiquement, une interface est représentée par un cercle.

d. ***Les cas d'utilisation :*** Formalisés par Ivar Jacobson, les cas d'utilisation décrivent sous la forme d'action et de réaction de comportement d'un système du point de vue de l'utilisateur et permettent de définir les limites du système déclenchée en réponse entre le système et l'environnement. C'est l'image de fonctionnalité du système déclenchée en réponse à la stimulation d'un acteur externe, les cas d'utilisation sont représentés par des ellipses contenues par le système.

III.2. Les éléments comportementaux :

Ils sont les parties dynamiques des modèles UML, ils comprennent les interactions et les automates à états finis.

a. ***Les interactions :*** Une interaction exprime le comportement qui résulte de la collaboration d'un groupe d'instances. Elle peut être visualisée selon le point de vue du temps (diagramme de séquence) ou selon le point de vue de l'espace (diagramme de collaboration). La figure suivante schématise une interaction visualisée selon le point de vue du temps.

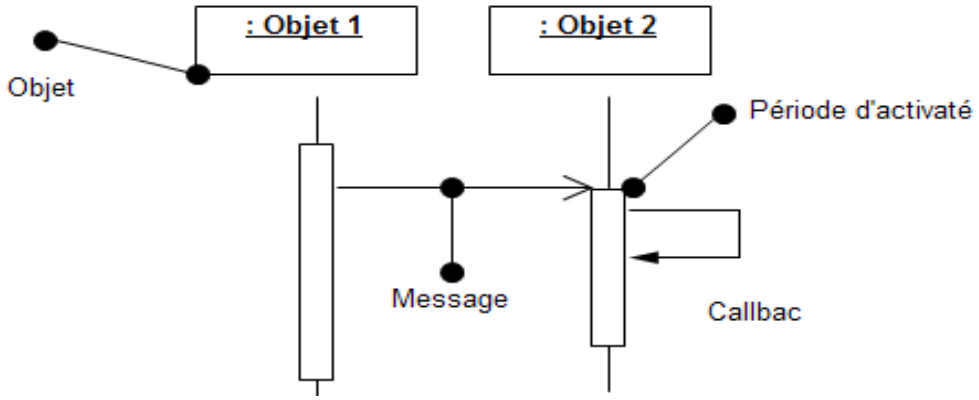


Figure A.3 : Schéma représentatif d'une interaction.

III.3.éléments de regroupement :

Ils sont les parties organisationnelles des UML, ils comprennent les paquetages.

- ***Les paquetages :***

Ils regroupent les trois types d'éléments précédents, ils sont purement conceptuels c'est-à-dire qu'ils n'existent que lors de la phase développement. Ils offrent un mécanisme général pour la partition des modèles et le groupement des éléments. Un paquetage est représenté en général par un dossier étiquette, contient seulement son nom, mais parfois son contenu.

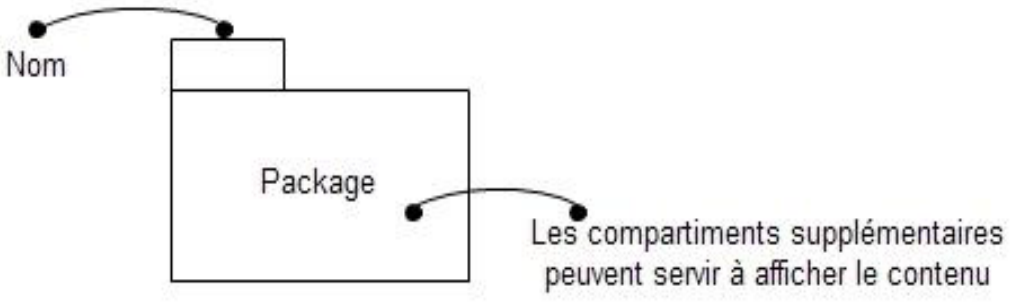


Figure A.4 : Schéma représentatif d'un paquetage.

III.4. Les éléments d'annotation :

Ils sont les parties explicatives des modèles UML ils comprennent les notes.

➢ **Les notes :** Ce sont les commentaires qui sont attachés à un ou plusieurs éléments de modélisation. Une note est représentée par un rectangle qui contient un commentaire textuel ou graphique. Comme le montre la figure suivante :

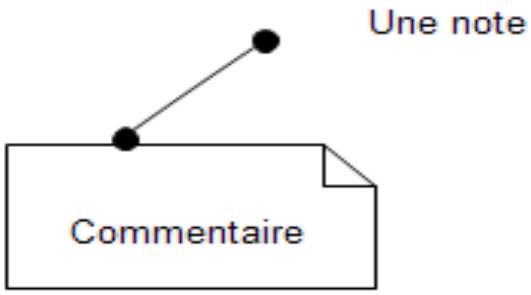


Figure A.5 : Schéma représentatif d'une note.

III.5 Les relations :

➢ ***Dépendance :*** Elle représente un lien de dépendance entre deux éléments de la modélisation et dont la modification d'un élément (élément indépendant) peut affecter la sémantique de l'autre élément (élément dépendant). Elle est représentée par un trait en pointillé.

Figure A.6 : Schéma représentatif d'une relation de dépendance.

➢ ***Association :*** Elle représente une relation entre classe d'objets. Elle exprime une connexion sémantique bidirectionnelle entre les classes. L'extrémité d'une association est appelée rôle, chaque rôle porte une indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe.

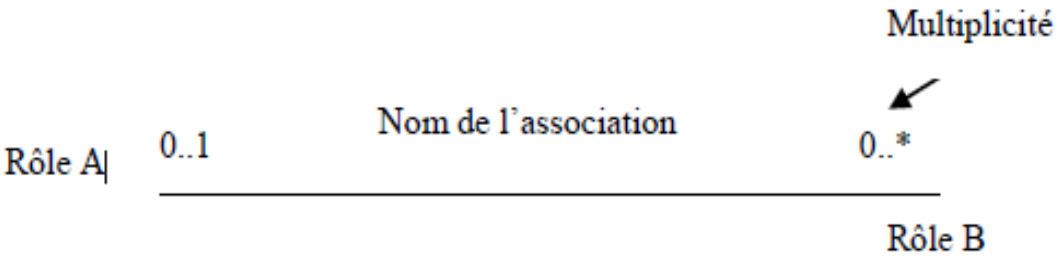


***Figure A.7*: Schéma représentatif d'une association**

- ***La généralisation :*** C'est une relation de spécification/généralisation entre une classe et une ou plusieurs autres classes partageant un sous-ensemble commun d'attributs et/ou de méthodes. UML emploie le terme de généralisation pour désigner la relation de classification entre un élément plus général (Parent) et un élément plus spécifique (enfant). La généralisation signifie « est une sorte de », elle peut s'appliquer aux classes, aux cas d'utilisation, et aux paquetages.

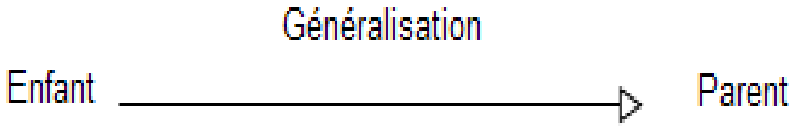


Figure A.8 : Schéma représentatif d'une généralisation.

III.6 Extensibilité :

UML comporte trois (03) mécanismes qui permettent d'étendre la syntaxe et la sémantique du langage.

- **Les stéréotypes :**

Les stéréotypes représentent de nouveaux éléments de modélisation, ils constituent un moyen de classer les éléments de la modélisation et facilitent l'élaboration du méta-modèle d'UML.

Ils s'appliquent principalement aux classes et rendent possible l'identification d'une typologie de classe souvent nécessaire lorsqu'on manipule un grand nombre de classes. Le nom du stéréotype est indiqué entre guillemets.

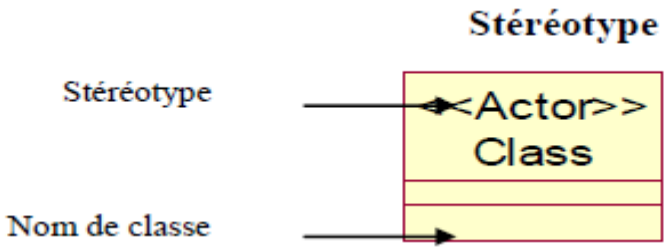


***Figure A.9* :** schéma représentatif d'une classe stéréotypée.

➢ **Les contraintes :** Une contrainte est une note ayant une valeur sémantique particulière pour un élément de la modélisation, elle s'écrit entre accolades {}, elle peut concerner plusieurs éléments

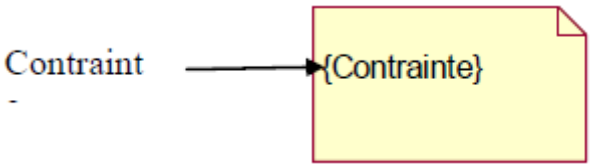


***Figure A.10* :** Schéma représentatif d'une contrainte

IV. Les diagrammes d'UML :

IV.1. Définition d'un diagramme UML :

Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle. C'est une perspective du modèle, chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis). Un type de diagramme UML véhicule une sémantique précise (un type de diagramme offre toujours la même vue d'un système).

Combinés les différents types de diagramme UML offre une vue complète des aspects statiques et dynamiques d'un système.

➢ Les caractéristiques d'un diagramme UML:
- Les diagrammes UML supportent l'abstraction. Leur niveau de détail caractérise le niveau d'abstraction du modèle;
- La structure des diagrammes UML la notation graphique des éléments de modélisation sont normalisées.

IV.2. Types de diagrammes : [51]

UML repose sur neuf types de diagrammes à savoir :

1. Le diagramme de classe : Diagramme structurel qui montre un ensemble de classes, d'interface, de collaboration ainsi que leurs relations.
2. Le diagramme d'objets : Diagramme structurel qui montre un ensemble d'objets ainsi que leurs relations.
3. Le diagramme de cas d'utilisation : Diagramme comportemental qui montre un ensemble de cas d'utilisation et d'acteurs ainsi que leurs relations.

➢ **Un acteur :** Un acteur représente un ensemble de rôles joués par les utilisateurs des cas d'utilisation en interaction avec ces cas d'utilisation. En règle générale, un acteur représente un rôle qu'un homme, une machine ou même autre système joue avec le système.

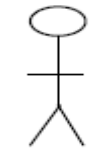


***Figure A.11* :** Représentation graphique d'un acteur

UML définit trois (03) types de relations pour le diagramme de cas d'utilisation.

- **La relation de communication :** Elle est signalée par une flèche entre l'acteur et le cas d'utilisation. Comme la montre la figure suivante :

***Figure A.12*:** Représentation d'un déclenchement d'un cas d'utilisation par un acteur

- **La relation d'utilisation :** Une relation d'utilisation entre cas d'utilisation signifie qu'une instance du cas d'utilisation source comprend également le comportement décrit par le cas d'utilisation destination. La figure suivante montre une relation d'utilisation :

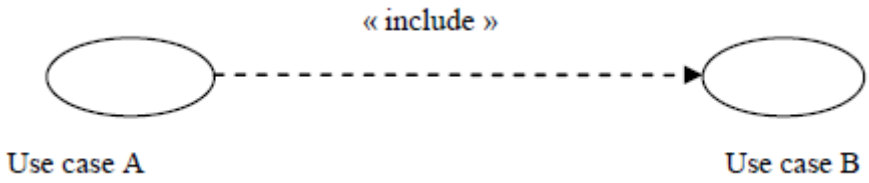


***Figure A.13*:** Représentation de la relation d'utilisation ou d'inclusion.

- **Relation d'extension :** Une relation d'extension entre cas d'utilisation signifie que le cas d'utilisation source étend le comportement du cas d'utilisation destination.

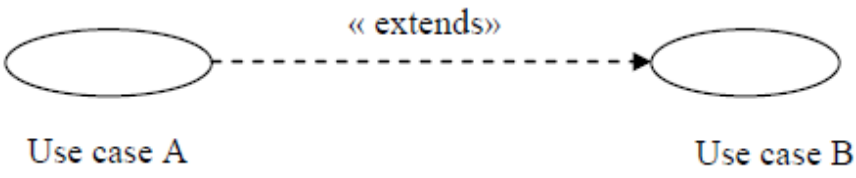


Figure A.14 : Représentation de la relation d'extension.

4. Le diagramme de séquences : Diagramme comportemental qui montre que une interaction et met l'accent sur l'ordre chronologique des messages.
5. Le diagramme de collaboration : Diagramme comportemental qui montre une interaction et met l'accent sur l'organisation structurelle des objets qui envoient et reçoivent des messages.
6. Le diagramme d'états-transitions : Diagramme comportemental qui montre un automate à états finis et met l'accent sur le comportement d'un objet ordonnancé par les événements.
7. Le diagramme d'activité : Diagramme comportemental qui montre un automate a états finis et met l'accent sur le flot d'une activité à l'autre.
8. Le diagramme de composante : Diagramme structurel qui montre un ensemble de comportements ainsi que leurs relations.
9. Le diagramme de déploiement : Diagramme structurel qui montre un ensemble de nœuds ainsi que leurs relations.

V. Extension UML pour les applications web : [51]

Une extension d'UML est définie par de nouveaux stéréotypes, étiquettes et contraintes. En les combinant, on peut créer de nouvelles briques de base que l'on pourra utiliser dans le modèle.

- Un stéréotype est une extension de vocabulaire d'UML. Il permet d'associer une nouvelle signification à un élément du modèle. Les stéréotypes peuvent être appliqués à presque tous les éléments du modèle et sont habituellement représentés par une chaîne de caractères entre guillemets (« »). On peut aussi les représenter par une icône.

– Une étiquette est une extension des propriétés d'un élément du modèle. La plupart des éléments du modèle possèdent des propriétés ; les classes, par exemple, possèdent, entre autre, un nom, une visibilité et une persistance. L'étiquette est la définition d'une nouvelle propriété d'un élément du modèle. Elle est représentée, dans un diagramme, par une chaîne de caractères entre chevrons (<>).

Une contrainte est une extension de la sémantique d'UML. La contrainte édicte une règle que le modèle doit vérifier pour être qualifié de « bien formé ». Les contraintes sont représentées par des chaînes de caractères entre accolades ({}). Une extension d'UML comporte une brève introduction, puis la liste des stéréotypes, étiquettes et contraintes avec leur description. Une extension contient aussi un ensemble de règles qui garantissent la cohérence sémantique du modèle

V.1. Description des stéréotypes:

V.1.1. Les Classes :

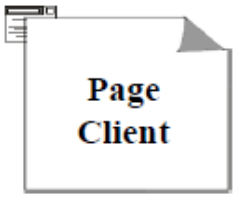
- **Page serveur « Server page » :**

Icône :

Description : Représente une page Web possédant des scripts exécutés par le serveur et qui interagissent avec des ressources du serveur telles que les bases de données.
Contraintes : Les pages serveur ne peuvent avoir de relations qu'avec des objets sur le serveur.
Etiquettes : Moteur de script qui peut être un langage ou le moteur qui doit être utilisé pour exécuter ou interpréter cette page.

- **Page client « Client page » :**

Icône :

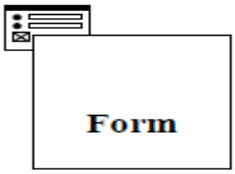


Description : Une instance d'une page client est une page Web formatée en HTML.
Les pages clients peuvent contenir des scripts interprétés par les navigateurs lorsque celles-ci sont restituées par ces derniers. Les fonctions des pages clients correspondent aux fonctions des scripts de la page web.
Contraintes : Aucune.
Étiquette :

- Titre (Title) : Titre de la page tel qu'il est affiché par le navigateur.
- Base (Base) : URL de base pour référencer les URL relatives.
- Corps (Body) : Eensemble des attributs de la balise <body>, qui définit des caractéristiques par défaut du texte et de l'arrière-plan.

- **Formulaire « Form »:**

Icône :



Description : Une classe stéréotypée « form » est un ensemble de champs de saisie faisant partie d'une page client. A une classe formulaire correspond une balise HTML <form>. Les attributs de cette classe correspondent aux éléments de saisie d'un formulaire HTML (zone de saisie, zone de texte, boutons d'option, case à cocher…etc.). Un formulaire n'a pas

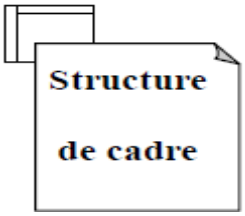
d'opérations, toute opération qui interagit avec le formulaire appartient à la page qui la contient.

Contraintes : Aucune.

Etiquettes : GET ou POST – Méthodes utilisées pour soumettre les données à l'URL de l'attribut action de la balise HTML <form>.

- **Structure de cadres « Frameset » :**

Icône :



Description : Une structure de cadres est un conteneur de plusieurs pages Web. La zone d'affichage rectangulaire est devisée en cadres rectangulaires inscrits. A chaque cadre peut être associé un nom unique de cible (Target). Le contenu d'un cadre peut être une page Web ou une structure de cadre. Une structure de cadre est une page client qui peut posséder des opérations et des attributs.

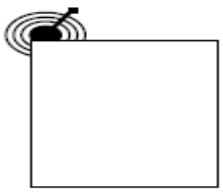
Contraintes : Aucune

Etiquettes :

- Rangées (rows) : valeur de l'attribut rows de la balise HTML <framset>. C'est une chaîne de pourcentages séparés par des virgules, définissant les hauteurs relatives des cadres.
- Colonnes (cols) : valeur de l'attribut cols de la balise HTML <frameset>. C'est une chaîne de pourcentages séparés par des virgules, définissant les largeurs des cadres.

- **cible « Target » :**

Icône :



Description : Une cible est une zone nommée dans la fenêtre du navigateur dans laquelle des pages Web peuvent être affichées. Le nom de la classe stéréotypée est celui de la cible. Habituellement, une cible est le cadre d'une structure de cadre définie dans une fenêtre ; cependant, une cible peut être une toute nouvelle instance de navigateur : une fenêtre. Une association « targeted link » spécifie la cible où une page Web doit être affichée.

Contraintes : Pour chaque client du système, le nom de la cible doit être unique. Par conséquent sur un même client, il ne peut exister qu'une seule instance d'une même cible.

Etiquettes : Aucune.

V.1.2. Les associations :

- **Lien « Link » :**

Icône: aucune.
Description : Un lien est un pointeur d'une page client vers une autre page. Dans un diagramme de classes, un lien est une association entre une page client et une autre page client ou une page serveur. A un lien correspond une balise ancre HTML.
Contraintes : aucune.
Etiquettes : *Paramètres* : liste des noms de paramètres qui doivent être passés avec la demande de la page liée.

- **Lien cible « Targeted link »:**

Description : Similaire à une association lien. Un lien cible est un lien dont la page associée est affichée dans une cible. A un lien cible correspond une balise ancre HTML, dont l'attribut target prend la valeur de la cible.
Contraintes : Aucune.
Etiquettes :

- Paramètres : Liste des noms de paramètres qui doivent être passés avec la demande de la page liée.
- Nom de la cible (target name) : Nom de la cible où la page vers laquelle pointe le lien qui doit être affichée.

- **Contenu de cadre « Frame content » :**

Description : Une association contenue de cadre est une association d'agrégation qui traduit l'appartenance d'une page ou d'une cible à un cadre. Une association contenue de cadre peut aussi pointer vers une structure de cadre, aboutissant dans ce cas, à des cadres imbriqués.
Contraintes : Aucune.
Etiquettes :

- *Rangée (Row)* **:** Entier qui indique la rangée du cadre dans la structure de cadre auquel
- appartient la page ou la cible associée.
- *Colonne (Col)* : Entier qui indique la colonne du cadre dans la structure de cadre auquel appartient la page, ou la cible associée.

- **Soumet « Submit » :**

Description : Submit est une association qui se trouve toujours entre un formulaire et une page serveur. Les formulaires soumettent les valeurs de leurs champs au serveur, par l'intermédiaire de pages serveur, pour qu'il les traite.
Contraintes : Aucune.
Etiquettes : Paramètres : Une liste des noms de paramètres qui doivent être passés avec la demande de la page liée.

- **Construit « Build » :**

Description : La relation « Build » est une relation particulière qui fait le pont entre les pages client et les pages serveur. L'association « Build » identifie quelle page serveur est responsable de la création d'une page client. C'est une relation orientée, puisque la page client n'à pas connaissance de la page qui est à l'origine de sont existence.

Une page serveur peut construire plusieurs pages client, en revanche, une page client ne peut être construite que par une seule page serveur.

Contraintes : Aucune.

Etiquettes : Aucune.

- **Redirige « Redirect » :**

Description : Une relation « Redirect », est une association unidirectionnelle avec une autre page web, elle peut être dirigée à partir d'une page client ou serveur ou vers une page client ou serveur.

Contraintes : Aucune.

Etiquettes : Aucune.

V.1.3. Les attributs :

- **Elément de saisie « input element » :**

Description : Un élément de saisie correspond à la balise <input> d'un formulaire HTML, cet attribut est utilisé pour saisir un mot ou une ligne. Les étiquettes associées à cet attribut stéréotypé, correspondent aux attributs de la balise <input>. Les attributs obligatoires de la balise HTML <input> sont renseignés de la manière suivante : l'attribut « name » prend la valeur du nom de l'élément de saisie et l'attribut « value » prend celle de sa valeur initiale.

Contraintes : Aucune.

Etiquettes :

- Type (Type) : Le type de l'élément de saisie : texte, numérique, mot de passe, case à cocher, bouton d'option, bouton « submit » ou bouton « reset ».
- Taille (size) : Définit la largeur visible allouée à l'écran en caractères.
- Longueur Max (Maxlength) **:** Nombre maximal de caractères que peut saisir l'utilisateur.

- **Sélection d'éléments « Select element » :**

Description : C'est un contrôle de saisie employé dans les formulaires, il permet à l'utilisateur de sélectionner une ou plusieurs valeurs dans une liste. La plupart des navigateurs restituent ce contrôle par une liste d'options ou une liste déroulante.

Contraintes : Aucune.

Etiquettes :

- Taille (Size) : Définit le nombre d'éléments qui doivent être affichée simultanément.
- Multiple (Multiple) : Valeur booléenne qui indique que plusieurs éléments peuvent être sélectionnés conjointement.

➢ **Zone de texte « text area element » :**

Description : C'est un contrôle de saisie, employé dans les formulaires, qui permet l'écriture de plusieurs lignes de texte.

Contraintes : Aucune.

Etiquettes

- Ligne (rows) : Nombre de lignes de texte visibles.
- Colonnes (cols) : Largeur visible du texte en largeurs de caractères moyennes.

V.2. Règles de cohérence

– Réalisation de composant : les composants pages Web peuvent réaliser les classes stéréotypées « server pge », « client page » ou « frameset ».

– Généralisation : tous les éléments de modélisation dans une même généralisation doivent être du même stéréotype.

– Association : une page client peut avoir au plus une relation « build » avec une page serveur, par contre une page serveur peut avoir plusieurs relations « build » avec plusieurs pages client. En plus des combinaisons standards d'UML sont permises les combinaisons de stéréotype présentées au tableau.

A: / De :	« client page »	« serveur page »	« frameset »	« target »	« form »
« client page »	« link » « target link » « redirect »	« link » « target link » « redirect »	« link » « target link » « redirect »	« dependency »	« aggregation »
« serveur page »	« build » « redirect »	« redirect »	« build » « redirect »		
« frameset »	« frame content »		« frame content »	« frame content »	
« target »					
« form »	« aggregate by »	« submit »			

Tableau 1 : Combinaison valides d'association de stéréotypes.