Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté de Science
Département d'Informatique

Mémoire de fin d'études
Pour l'obtention du diplôme de Master en Informatique

Option : Système d'Information et de Connaissance(SIC)

Thème

Conception et développement d'une application médicale distribuée à l'aide d'un composant Entreprise Java beans (EJB)

Réalisé par :

- **Larouci Bouchra**
- **Meddahi Imene**

Présenté le 3 Juillet 2017 devant le jury composé de MM .

- Mme El Yebdri Zineb (Présisdente)
- Mr Belabed Amine (Examinateur)
- Mr Bennamar Abdelkrim (Encadreur)

Année universitaire: 2016-2017

Remerciements

A l'issue de ce travail, nous voulons d'abord remercier Allah de nous guider et donner la force, le courage et la patience pour achever ce modeste travail.

Nous tenons à exprimer nos profondes gratitudes et sincères remerciements à notre encadreur Mr. Bennamar pour avoir accepté diriger ce travail et pour leurs aides et leurs orientations.

Nous tenons à remercier vivement les membres de jury qui ont fait l'honneur d'accepter de participer à notre soutenance de ce mémoire.

Chaleureux remerciements vont à tous nos enseignants au cours de notre formation au fil des années.

Ce travail n'aurait pas été possible sans le soutien affectueux de plusieurs personnes, nous trouvons submergés en leur offrant tous nos remerciements à dédier ce travail pour eux.

Dédicace

Je tiens à dédier ce modeste travail à la lumière de ma vie: mes chers parents, mon père « Abderrahmane », et ma mère « Khadra », pour leur plus grand amour, soutien, encouragement de la patience et de l'aide continue pendant mes années d'études.

Ce travail est également dédié: À mes sœurs douces Zahia et Bouchra
Pour mon cher frère « Mohamed El Amine »
A mes chers grands Parents rabi yerhamhom.
Pour tous mes oncles et mes tantes, Pour toute la famille MEDDAHI et la famille HAMDANE.

A touts mes amis en qui j'ai toujours trouvé le soutien et le réconfort.

JE désire dédier ce travail aussi à tous mes amies dans le campus, et à tous ceux qui ont contribué à la réalisation de ce travail.

A mon binôme, ma soeur, et mon amie Bouchra

Pour tous ces gens aimables je suis très reconnaissante.

A qui celui qui demeure dans mon cœur : l'ALGÉRIE

Meddahi Imene

Dédicace

Toutes les lettres ne sauraient trouver les mots qu'il faut, tous ces mots ne sauraient exprimer la gratitude, l'amour, le respect, et la reconnaissance à mes très chers parents qui m'ont offert sans condition leur soutien morale et financier et à qui je dois ce travail.

A mes grands-parents : 'Tafilala khaira' et 'Mohamed' que Dieu les Touts Puissant les gardent en bonne santé.

A mes sœurs : Ikram et Farah.

A mes chères oncles : Mohamed et Abdel Kader pour tous leurs encouragements.

A mes tantes : Samira, Radia et surtout Rabia.

Et surtout mes adorables cousins : Diaa Din et Abdel Rahmen.

A toute ma famille : « LAROUCI »

A tous mes amis avec lesquels j'ai partagé mes moments de joie, bonheur et folie Mohamed El Amine, Ahmed Issam.

A mes cousines : Ibtissem Chourouk …

Et à toutes ces personnes spéciales qui ont, durant toutes ces années, été mes compagnons de tous les jours et une deuxième famille pour moi, ceux-là qui n'ont cessé de partager avec moi leur sympathie, leur chaleur, leur estime et leur indulgence : Amina wahiba ...

A tous les membres de ma promotion SIC, à tous mes professeurs.

A mon encadreur " Mr Benammar" pour son soutien, sa compréhension, sa patience ces encouragements et ces aides.

Et enfin à mon binôme « Meddahi Imene » avec laquelle j'ai partagé de belles années d'études et avec qui j'ai eu l'honneur de les finir.

Je dédie ce modeste travail.

Larouci Bouchra

TABLE DES MATIERES

LISTE DES FIGURES

INTRODUCTION GENERALE

Introduction générale

L'informatisation est le phénomène le plus important de notre époque. Elle s'immisce maintenant dans la plupart des objets de la vie courantes .Le phénomène informatique qui n'est plus exclusivement un outil de bureautique, s'est progressivement et rapidement répandu dans tous les domaines.

De nos jours, dans le domaine de la santé, l'informatique prend petit à petit une place importante et grandissante.

Dans ce cas nous avons utilisé cette technologie de l'informatisation pour développer une application médicale.
C'est donc en essayant de rénover la gestion d'une clinique que l'informatisation de la pratique médicale s'impose.
Ce système accompagne ses utilisateurs dans ses pratiques quotidiennes pour améliorer ses services à bonnes gestions , afin d'augmenter la fiabilité, l'efficacité de l'effort humain et faciliter les tâches pénibles au sein du clinique.
D'ou notre travail revient à la conception et a la réalisation d'un Système d'informations de Gestion d'une clinique médicale.

Nous présenterons dans le premier l'architecture à base de composants : CORBA, .Net, J2EE sur lequel nous nous sommes focalisés notre architecture pour réaliser notre application.
Dans le deuxième chapitre, on présente les composants entreprises java beans.
Le troisième chapitre donne un aperçu détaillé de la conception du système conçu .
Le dernier chapitre présente les outils qui ont servi au développement ainsi que la réalisation de l'application et l'exposition de quelques interfaces du système réalisé.
Enfin, nous finirons par une conclusion en mettant le point sur l'importance des technologies utilisées.

1. ARCHITECTURE A BASE DE COMPOSANTS

1.1 Introduction

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

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

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

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

Les bénéfices attendus de l'utilisation des composants sont a priori clairs : il s'agit essentiellement de réduire les coûts de production des logiciels, mais aussi d'en réduire les coûts de maintenance et d'en favoriser la réutilisation

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

1.2 CORBA

1.2.1 Définition

CORBA (Common Object Request Broker Architecture) est une architecture proposée et maintenue par l'OMG (Object Management Group) pour permettre l'intégration d'une grande variété de systèmes hétérogènes distribués orientés objet.

CORBA est conçue pour supporter des applications distribuées orientées objet et développées selon le modèle client-serveur.
On dit aussi que CORBA est un bus réparti (à ne pas confondre évidemment avec la notion de bus physique utilisée dans les réseaux de transmission).

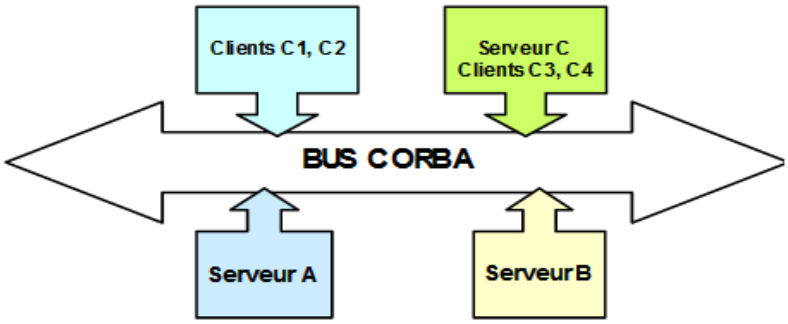

Figure1.1 : Bus CORBA [1]

CORBA a été conçue pour développer des applications de la façon la plus simple possible, même si développer des applications distribuées n'est pas toujours facile. Transparence, portabilité et interopérabilité sont les principes de base de la spécification de CORBA. Quand le client fait une requête, il ne voit qu'un appel local, tous les mécanismes impliqués dans l'appel sont transparents au client.
Un des avantages les plus importants de la spécification CORBA est de faciliter le développement des applications distribuées en évitant au développeur des difficultés concernant les mécanismes de communication, la location des applications, les clients et serveurs, le protocole de communication à utiliser ou le format de message à utiliser. [1]

1.2.2 Développement CORBA

ORB : Object Request Broker

L'Object Request Broker (ORB), est un composant fondamental de l'architecture CORBA ; sa mission est de faciliter la communication entre objets : il est chargé d'envoyer les requêtes aux objets et de retourner les réponses au client qui les a invoqués par un processus de sérialisation.

IDL : Interface Definition Language

Le langage de définition des interfaces ou IDL (Interface Definition Language), est utilisé pour définir les interfaces entre les composants d'une application et permettre l'interopérabilité entre différentes technologies.

1.2.3 Déroulement de requête sous CORBA

Le client n'a pas besoin de connaître :

- la localisation de l'objet invoqué,
- le langage avec lequel a été programmé l'objet,
- le système d'exploitation sur lequel est implanté l'objet,
- la façon dont l'objet est implantée par le serveur.

Le client doit connaître seulement l'interface d'accès à l'objet et la référence de l'objet. Les interactions entre les applications sont matérialisées par des invocations, à distance ou en local, des méthodes(ou opérations) des objets.
Pour pouvoir faire une requête, le client peut utiliser l'interface d'invocation dynamique ou un stub (souche) produit pendant la compilation d'un script IDL. L'implémentation de l'objet et le client peuvent dialoguer avec l'ORB à travers les interfaces de l'ORB.

Les interfaces pour les objets peuvent être définies de deux façons :
Statique : en utilisant le langage de définition d'interfaces (IDL). Le langage IDL permet de définir les opérations sur les objets et les paramètres des opérations.
Dynamique : où les interfaces peuvent être stockées dans un dépôt d'interfaces. Les composantes des interfaces sont représentées comme objets afin de permettre leur accès pendant l'exécution.

Pour pouvoir faire une requête, le client doit connaître la référence de l'objet, le type et l'opération à exécuter. Une fois cette information obtenue, le client peut initialiser la requête. La requête se fait en appelant les routines représentées par les stubs ou en les construisant de manière dynamique. Un stub est spécifique à un objet. [1]

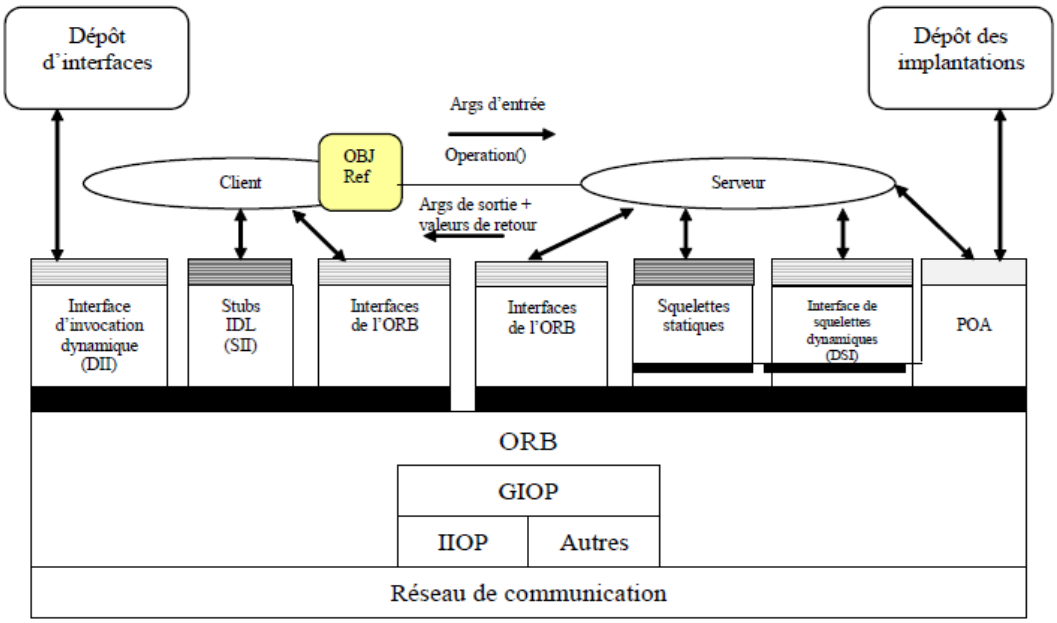

Figure 1.1 : Structure de CORBA [1]

1.3 Dot Net

1.3.1 Définition

Microsoft .NET est une architecture logicielle destinée à faciliter la création, le Déploiement des applications en général, mais plus spécifiquement des applications Web ou des services Web. Cette architecture logicielle concerne aussi bien les clients que les serveurs qui vont dialoguer entre eux à l'aide d'XML. Elle se compose de quatre principaux éléments : [2]

- un modèle de programmation qui prend en compte les problèmes liés aux Déploiements des applications (gestion de version, sécurité) ;
- des outils de développement dont le cœur est constitué de Visual Studio .NET
- un ensemble de systèmes serveurs représentés par Windows Server 2003, SQL Server ou Microsoft Biztalk Server qui intègrent, exécutent et gèrent les services Web et les applications ;
- un ensemble de systèmes clients représenté par Windows XP, Windows CE ou Microsoft Office 2003.

1.3.2 Les langages de programmation

Le langage de prédilection de l'environnement .NET est le langage C# (prononcé C Sharp), langage inventé par l'un des concepteurs de Delphi et J++.

D'autres langages peuvent être utilisés (il en existe plus de 20): C# , C++/CLI , F# , J# , Windows Power Shell , JScript.NET , IronPython , IronRuby , Managed Extension for C++ , Managed Jscript , VBx , VB.NET , A# , Ja.NET , Boo , Cobra , Component Pascal , IronLips , L# , Mondrian , Nemerle , Oxygene , P# , Phrogram , Power Builder , Delphi .NET , Fortran .NET

Ces langages doivent proposer des concepts orientés objets et un typage fort (ou une émulation de ces mécanismes). Ils peuvent donc avoir connu des modifications importantes par rapport à leurs versions originales (lorsqu'elles existent).

1.3.3 Architecture

CLR :

Le composant central de l'architecture .NET est le CLR (Common Language Runtime) qui est un environnement d'exécution pour des applications réparties écrites dans des langages différents. En effet, avec .NET on peut véritablement parler d'interopérabilité entre langages. Cette interopérabilité s'appuie sur le CLS (Common Language Spécification) qui définit un ensemble de règles que tout compilateur de la plate-forme doit respecter.

MSIL et JIT :

Pour permettre la compilation de différents langages sur la plate-forme .NET, Microsoft a défini une sorte de langage d'assemblage, indépendant de tout processeur baptisé MSIL (Microsoft Intermediate Language). Pour compiler une application destinée à .NET, le compilateur concerné lit le code source dans un langage respectant les spécifications CLS, puis génère du code MSIL.

Pour compiler à l'exécution le code MSIL, la plate-forme .NET utilise un compilateur à la volée (JIT Compiler - just-in-time compiler) qui place le code produit dans un cache. [3]

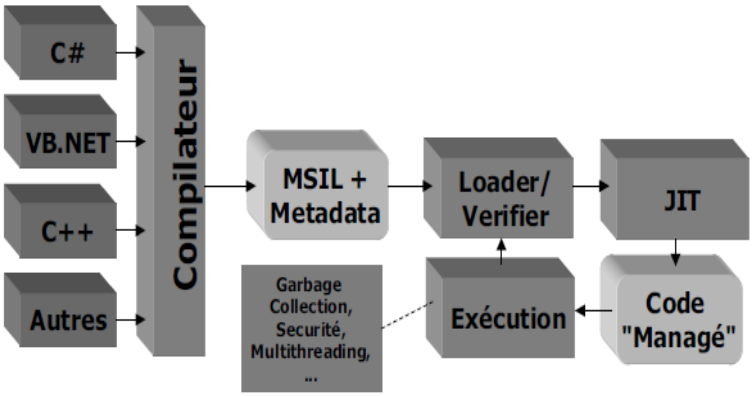

Figure 1.2 : Principe de fonctionnement du CLR [4]

1.3.4 Le déploiement d'applications .NET

Les applications .NET sont organisées sous la forme d'un **Assembly**.

Outre les différents fichiers MSIL, les assemblies comportent un fichier **Manifest** décrivant les caractéristiques de déploiements (sécurité, version, dépendances, …).

Les assemblies peuvent être signés, déployées de manière partagée ou privée. [4]

1.4 J2EE

Le langage Java se décline autour de 3 briques :

- Java 2 Standard Edition (J2SE) : vise les postes clients.
- Java 2 Micro Edition (J2ME) : cible les terminaux portables (téléphonie, PDA).
- ***Java 2 Entreprise Edition (J2EE) : vise les applications d'entreprise*.**

1.4.1 Définition J2EE

Conscient de l'intérêt des architectures multi-tiers pour le développement d'applications d'entreprises, la société SUN Micro Systems a proposé, dès 1999, une déclinaison de son SDK Java (Software Development Kit) baptisé J2EE (Java 2 Enterprise Edition).
J2EE est un ensemble de spécifications (et non pas un produit) qui, en respectant une architecture multi tiers, va décrire à la fois :

- *l'infrastructure de gestion des applications*
- *les API des services utilisées pour concevoir ces applications.*

1.4.2 Organisation d'une architecture J2EE

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

J2EE définit un modèle d'architecture multi niveaux basé sur des composants sépare très nettement les trois niveaux d'abstraction d'une application (présentation logique métier, accès aux données).

Cette architecture est organisée de la façon suivante :

Tiers client : il représente l'application cliente (applet, web browser, application java).

Tiers web : il représente le serveur web J2EE, il se charge de la partie présentation.

Tiers métier : il représente le serveur métier J2EE (serveur EJB) pour implémenter le logique métier de l'application.

Tiers données : il regroupe l'ensemble des données stockées dans le serveur de données (SGBD, LDAP…). [5]

1.4.3 Composants de l'architecture J2EE

La technologie EJB : c'est un standard de développement de composants métiers.

Servlet : les servelets sont des programmes Java qui s'intègrent à un serveur d'application Web pour fournir le traitement coté serveur des requêtes issues d'un navigateur web client : ils s'appuient sur un serveur web supportant la technologie Java serveur tel que Tomcat.

Il est possible d'utiliser un programme écrit dans n'importe quel langage pour envoyer des requêtes à une servlet ; cela veut dire que le client peut être une simple page HTML, une applet Java ou un programme écrit dans un autre langage que java. La servlet traite le requête et génère une sortie dynamique vers le client qui peut être une page HTML, XML ou encore un objet Java sérialisé.

Java Server Page (JSP) : la technologie JSP est une extension des servelets qui offre un moyen simplifié pour écrire des servelets ; elle combine du code Java et du code HTML dans une page

JSP. Toutefois, elle permet la gestion des pages web dynamique comme les ASP(Active Server Page), PHP(Personal Home Page), etc.… [5]

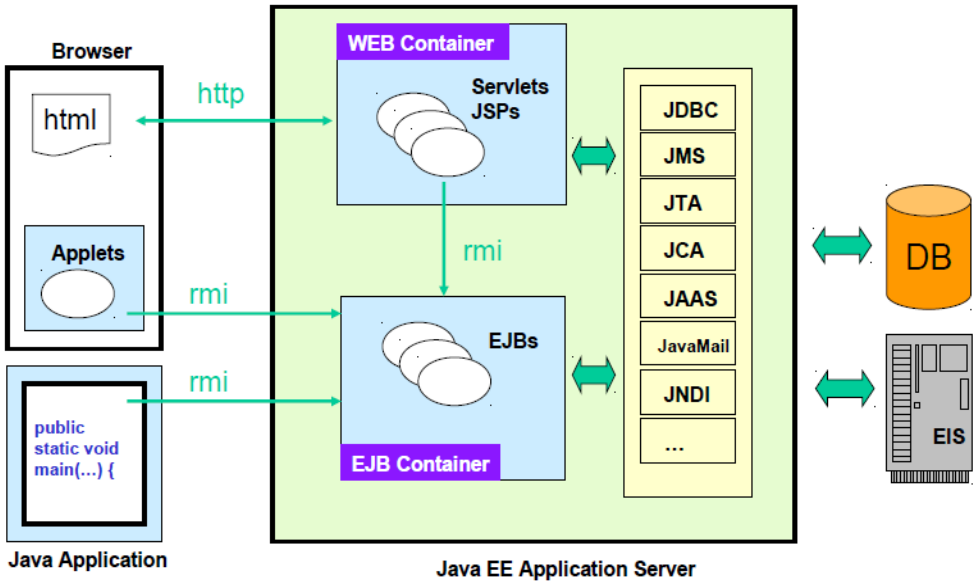

Figure 1.3 : Architecture J2EE[6]

1.4.4 Les Services de l'architecture J2EE

Remote Method Invocation-Internet Inter ORB Protocol (RMI-IIOP) :
C'est une extension de RMI, utilisé pour la communication entre les composants distribués.

Java Naming and Directory Interface(JNDI) : c'est une API permettant l'accès aux services de nommage, elle peut être utilisée pour différentes raisons comme la localisation d'un composant EJB ou de n'importe quelle ressource à travers le réseau.

Java Database Connectivity(JDBC) : c'est une API qui permettant l'accès aux bases de données relationnelles.

***Java Message Service(JMS)* :** c'est une API qui permet la communication entre composants par envoi de message.

***J2EE Connector Architecture(JCA)* :** elle offre un service d'accès à des informations d'entreprise existantes à partir d'application J2EE.

Java API for XML Parsing(JAXP) : c'est une API qui permet le traitement des documents XML.

Java Transaction API/Java Transaction Services(JTA/JTS) : c'est une API définissant des interfaces standards avec un gestionnaire de transactions.

JAVA-MAIL : elle permet d'envoyer des messages e-mails indépendamment du protocole.

JAVA IDL : permet l'implémentation de composants CORBA en java.

Java API for XML RPC(JAX-RPC) : c'est la technologie principale pour le développement des web services, elle définit deux modèles, l'un basé sur les servelets et l'autre sur les EJB.

Java Authentification and Authorization Service(JAAS) : elle prend en charge l'aspect sécurité des opérations.

Java Management Extension(JMX) : elle fournit des extensions permettant de développer des applications web de supervision d'applications. [5]

1.4.5 Outils de développement

Dans le monde J2EE, de nombreux outils de développement (IDE) existent depuis plusieurs années:

- Borland propose une version (limitée) gratuite de JBuilder.
- Dans le domaine du logiciel libre, IBM a initié un projet ambitieux d'IDE multi-langages baptisé Eclipse.
- Le projet NetBeans, initié par SUN, est concurrent du projet Eclipse.

1.4.6 Le Runtime

Les programmes Java sont compilés en un code intermédiaire baptisé **bytecode Java** (ou fichiers**.Class**). Ce code est celui d'une **machine virtuelle Java** (**JVM**).

Cette machine virtuelle Java est émulée par un logiciel (la JVM).

Il existe des JVM pour un grand nombre de platesformes. De plus, de nombreux browsers ont une JVM. [4]

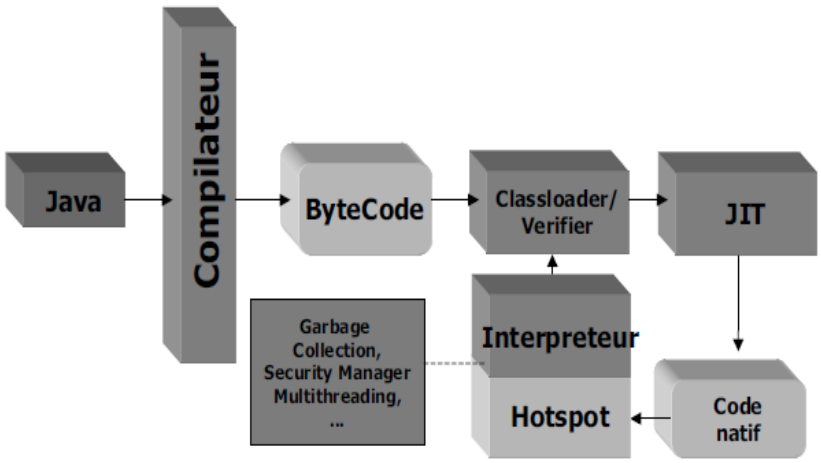

Figure 1.4 : Principe de fonctionnement du JVM [4]

1.5 Conclusion

Nous avons vu à travers ce chapitre ce qu'était l'architecture à base de composants et en quoi des architectures tels que Corba, .net, et enfin J2EE sur lequel nous nous sommes focalisé, intervenaient.

2. ENTREPRISE JAVA BEANS (EJB)

2.1 Introduction

Les systèmes à base de composants sont construits par assemblage de composants il va de soi que tout modèle de composants vise à définir une infrastructure permettant la communication aisée entre ces composants dans un environnement multi-tiers.
C'est dans cette perspective que le modèle EJB s'inscrit pour fournir un modèle de composants standards distribués, portables, sécurisés, transactionnels et interopérables.

Une des principales caractéristiques des EJB est de permettre aux développeurs de se concentrer sur les traitements orientés métiers car les EJB et l'environnement dans lequel ils s'exécutent prennent en charge un certain nombre de traitements tel que la gestion des transactions, la persistance des données, la sécurité, ...

Physiquement, un EJB est un ensemble d'au moins deux interfaces et une classe regroupées dans un module contenant un descripteur de déploiement particulier.

Le but des EJB est de faciliter la création d'applications distribuées pour les entreprises.

2.2 La présentation des EJB

Les Entreprise Java Bean ou EJB sont des composants serveurs donc non visuels qui respectent les spécifications d'un modèle éditées par Sun. Ces spécifications définissent une architecture, un environnement d'exécution et un ensemble d'API. .

Le respect de ces spécifications permet d'utiliser les EJB de façon indépendante du serveur d'applications J2EE dans lequel ils s'exécutent, du moment que le code de mise en œuvre n'utilise pas d'extensions proposées par un serveur d'applications particulier.

Les JavaBeans possèdent certaines caractéristiques comme la réutilisabilité, la possibilité de s'assembler pour construire une application, etc ...

Les EJB sont parfaitement adaptés pour être intégrés dans une architecture trois tiers ou plus. Dans une telle architecture, chaque tier assure une fonction particulière :

- le client « léger » assure la saisie et l'affichage des données
- sur le serveur, les objets métiers contiennent les traitements. Les EJB sont spécialement conçus pour constituer de telles entités.

- une base de données assure la persistance des informations

Les EJB s'exécutent dans un environnement particulier : le serveur d'EJB. Celui-ci fournit un ensemble de fonctionnalités utilisées par un ou plusieurs conteneurs d'EJB qui constituent le serveur d'EJB. En réalité, c'est dans un conteneur que s'exécute un EJB et il lui est impossible de s'exécuter en dehors.

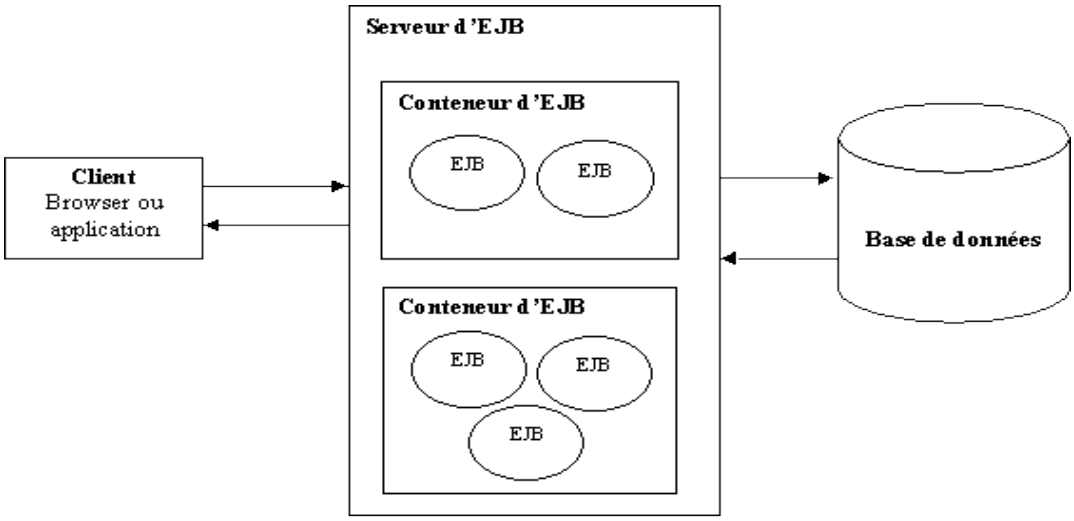

Figure 2.1 : architecture générale d'EJB[7]

Les composants EJB appelés entreprise beans (beans) sont déployables et peuvent être importés et chargés dans un serveur d'application.[7]

2.3 Interfaces EJB

Pour chaque EJB session, le développeur doit fournir une (ou 2) interface qui indique les méthodes de l'EJB que les clients de l'EJB pourront appeler

Les autres méthodes de l'EJB servent au bon fonctionnement de l'EJB

Un EJB session peut avoir un :

Interface locale :

Si l'EJB n'a qu'une seule interface locale, il ne peut être utilisé que par les classes qui sont dans le même container.

Le développeur peut ne fournir aucune interface ; en ce cas, une interface locale est Automatiquement créée, qui contient toutes les méthodes publiques de l'EJB.

Interface distante :

Indispensable si l'EJB peut être utilisé par des classes qui ne sont pas dans le même container (application distribuée)

Pour manipuler un EJB à travers une interface locale, le serveur d'application utilisera RMIIIOP, ce qui implique des performances moins bonnes

les paramètres et les valeurs de retour sont transmis par recopie des valeur (références pour un appel local). [8]

2.4 Les différents types d'EJB

Il existe deux types d'EJB : les beans de session (session beans) et les beans entité (les entity beans). Depuis la version 2.0 des EJB, il existe un troisième type de bean : les beans orientés message (message driven beans).

Les session beans peuvent être de deux types : sans état (stateless) ou avec état (stateful).

Les beans de session **sans état** peuvent être utilisés pour traiter les requêtes de plusieurs clients.

Les beans de session **avec état** ne sont accessibles que lors d'un ou plusieurs échanges avec le même client. Ce type de bean peut conserver des données entre les échanges avec le client.

Les beans entités assurent la persistance des données. Il existe deux types d'entity bean :

- Persistance gérée par le conteneur (CMP : Container Managed Persistence).
- Persistance gérée par le bean (BMP : Bean Managed Persistence).

Avec un bean entité CMP (container-managed persistence), c'est le conteneur d'EJB qui assure la persistance des données.

Un bean entité BMP (bean-managed persistance), assure lui-même la persistance des données grâce à du code inclus dans le bean. [7]

Message-Driven Bean : Introduits à partir de la norme EJB 2.0 (aujourd'hui en 3.0)

Similaire aux Session beans : représentent des verbes ou des actions, On les invoque en

leur envoyant des messages, souvent d'une autre application

Ex : message pour déclencher des transactions boursières, des autorisations d'achat par CB. [9]

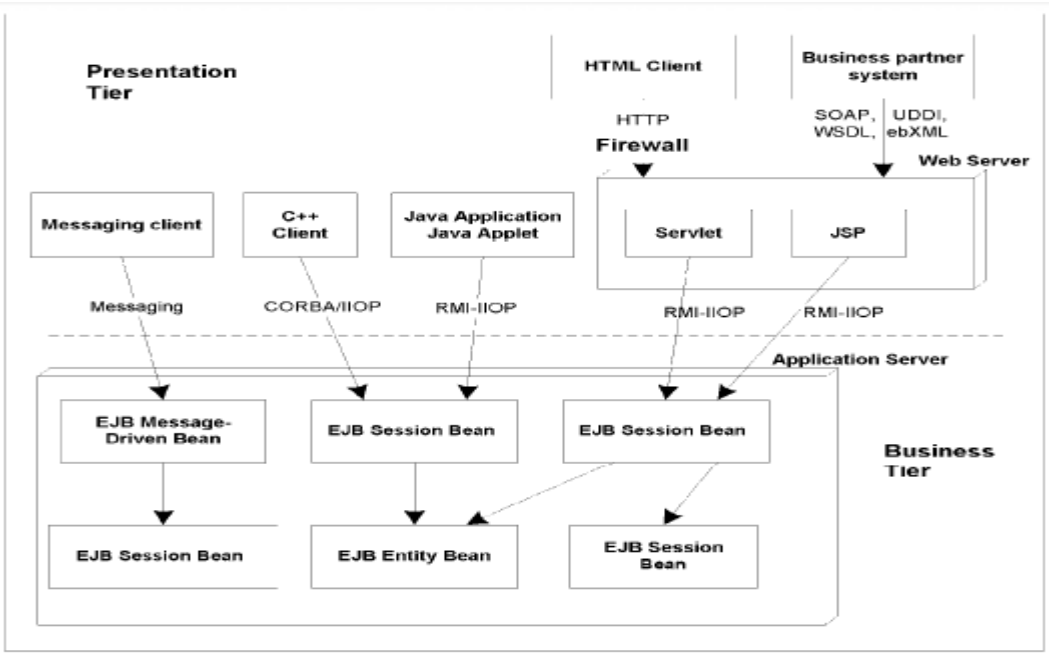

Figure 2.2 : les intéractions des clients avec les composants EJB

2.5 Les différents rôles d'EJB

Les spécifications des EJB définit six rôles qui sont utilisés pour réaliser les taches de développement et de déploiement ainsi que la définition du déploiement des composants du systèmes :

Le fournisseur d'entreprise bean : C'est le développeur qui est responsable de la création des composants EJB. Ces composants sont regroupés dans un fichier jar Spécial.

L'assembleur d'application : C'est l'architecte de l'application ; son rôle est de construire une application à partir d'un ensemble de composants.

L'assembleur d'application est le client des entreprise beans fournit par le fournisseur d'entreprise bean.

Le déployeur : Après que l'application eut été assemblée, elle doit être déployée sur un ou plusieurs serveurs d'applications. Le déployeur prend en considération :

a. La sécurité du déploiement par l'utilisation d'un parfeu et d'autres mesures de protection.

b. Le choix du matériel qui fournit le niveau requis de qualité de service.

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

L'administrateur système : c'est le rôle le plus important de l'ensemble du système : mettre en place et faire fonctionner. La gestion d'une application distribuée consiste à ce que les composants et les services soient tous configurés et interagissent ensemble correctement.

Le fournisseur du conteneur et du serveur EJB : Fournit un environnement d'exécution et les outils qui sont utilisés pour déployer, administrer et faire fonctionner les composants EJB.

Vendeur d'outil : Pour faciliter le développement de composants, il existe plusieurs IDE (Integrated Developement Environmrnt) permettant de construire, contrôler et maintenir les composants. Les IDE les plus populaires sont JBuilder, NetBeans et Eclipse. La plupart de ces outils permettent la modélisation des composants avec UML (Unified Modeling Language). [5]

2.6 Caractéristiques des EJB

- Largement utilisés par les entreprises : L'objectif principal des EJB est d'appliquer le principe « write once run anywhere » c'est-à-dire coder une seule fois, éxécuter n'importe ou. Ainsi, les entreprises utilisant les EJB seront bénéficiaires du fait que leur utilisation est très répandue. Elles peuvent produire donc des logiciels de haute qualité.
- Portabilité : Puisque la spécification des EJB est un standard, ils peuvent etre utiisés sans etre dépendants d'un fournisseur donné.
- Développement rapide d'application : Le développemnt des applications est plus rapide avec les EJB, grâce aux services middleware offerts tels que les transactions, la sécurité et le polling. [5]

2.7 Le développement d'un EJB

Concrètement un EJB est un groupe de deux interfaces accompagné d'au moins une classe dans un module contenant un descripteur de déploiement.

Quatre étapes sont nécessaires pour construire un EJB :

- Création de l'interface Home qui contrôle le cycle de vie de l'EJB .
- Création de l'interface Component qui contient la logique applicative
- Création du bean qui contient les méthodes.
- Création du deployment descriptor qui décrit l'EJB et les services dont il a besoin.

La gestion des transactions, la persistance des données et la sécurité sont directement prises en charge par les EJB. Les EJB communiquent entre eux en utilisant le JNDI (Java Naming Directory Interface).
La création d'un bean nécessite la création d'au minimum deux interfaces et une classe pour respecter les spécifications de Sun : la classe du bean, l'interface remote et l'interface home.
L'interface remote permet de définir l'ensemble des services fournis par le bean.

Cette interface étend l'interface EJBObject. Dans la version 2.0 des EJB, l'API propose une interface supplémentaire, EJBLocalObject, pour définir les services fournis par le bean qui peuvent être appelés en local par d'autres beans. Ceci permet d'éviter de mettre en œuvre toute une mécanique longue et coûteuse en ressources pour appeler des beans s'exécutant dans le même conteneur.

L'interface home permet de définir l'ensemble des services qui vont assurer la gestion du cycle de vie du bean. Cette interface étend l'interface EJBHome.

La classe du bean contient l'implémentation des traitements du bean. Cette classe implémente les méthodes déclarées dans les interfaces home et remote. Les méthodes définissant celles de l'interface home sont obligatoirement préfixées par "EJB".

L'accès aux fonctionnalités du bean se fait obligatoirement par les méthodes définies dans les interfaces home et remote. [7]

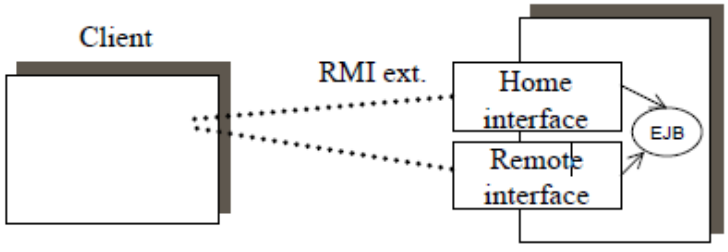

Figure 2.3 : architecture d'utilisation d'un EJB

2.8 Fournisseur de serveur/conteneur d'EJB

Le conteneur permet l'encapsulation d'un composant.
Il fournit des services de nommage, de gestion du cycle de vie, persistance, sécurité, transactionnel, ces services peuvent être demandés par appels de méthodes par le client. Ils seront réalisés par des méthodes propres au bean. [10]
Il existe plusieurs conteneurs d'EJB commerciaux mais aussi d'excellents conteneurs d'EJB open source notamment Glassfish, JBoss ou Jonas. Le conteneur d'EJB propose un certain nombre de services qui assurent la gestion : [7]

- du cycle de vie du bean
- de l'accès au bean
- de la sécurité d'accès
- des accès concurrents
- des transactions .

2.9 Les outils de développement

Plusieurs EDI (Environnement de Développement Intégré) open source permettent de développer et de tester des EJB notamment Eclipse et NetBeans. NetBeans est d'ailleurs celui qui propose le plus rapidement une implémentation pour mettre en œuvre la dernière version des spécifications relatives aux EJB.

2.10 Le déploiement des EJB

Un EJB doit être déployé sous forme d'une archive jar contenant un fichier qui est le descripteur de déploiement et toutes les classes qui composent chaque EJB (interfaces home et remote, les classes qui implémentent ces interfaces et toutes les autres classes nécessaires aux EJB).

Une archive ne doit contenir qu'un seul descripteur de déploiement pour tous les EJB de l'archive. Ce fichier au format XML doit obligatoirement être nommé ejb-jar.xml.

L'archive doit contenir un répertoire META-INF (attention au respect de la casse) qui contiendra lui-même le descripteur de déploiement.

Le reste de l'archive doit contenir les fichiers .class avec toute l'arborescence des répertoires des packages.

Le jar des EJB peut être inclus dans un fichier de type EAR. [7]

2.11 Processus de Développement, de Déploiement et d'Exécution

2.11.1 Développement d'un EB :

- Ecrire la classe Primary Key (pour un "entity bean")
- Ecrire la Home interface
- Ecrire la Remote interface
- Ecrire l'implémentation du bean
- Compiler ces classes et interfaces

2.11.2 Déploiement de l'EB :

- Construire le descripteur de déploiement (deploytool)
- Construire le fichier d'archive .ear
- Vérifier le fichier d'archive

2.11.3 Exécution de l'EB :

- Activer j2ee
- Exécuter le client qui peut être Une servlet (éventuellement dans un .war déployé) ou une application Java. [10]

2.12 Conclusion :

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

3. CONCEPTION DU SYSTEME

3.1 Introduction

Parmi les étapes les plus importantes de la programmation l'étape de conception, qu'on ne peut pas y dépasse et sans passer de cet étape on trouve des grandes erreurs. L'approche utilisée dans la conception de notre travail est l'approche objet. Elle consiste à obtenir un modèle informatique d'une collection d'éléments d'une partie du monde réel en un ensemble d'entités appelées objets. Cette approche présente plusieurs avantages dont les principaux sont : la modularité, l'extensibilité et la réutilisation.

Nous allons choisir pour notre application, la modélisation basée sur le langage UML et le processus de développement UP (Unified Process). Dans cette optique nous utiliserons les diagrammes de cas d'utilisation, de séquence, et de classe.

Parmi les multiples outils logiciels utilisés pour dessiner nos diagrammes, nous avons choisi StarUML par ce qu'il est libre et gratuit.

3.2 la méthodologie de conception

3.2.1 Présentation d'UML

UML (Unified Modeling Language) est un langage formel et normalisé en termes de modélisation objet. Son indépendance par rapport aux langages de programmation, aux domaines de l'application et aux processus, son caractère polyvalent et sa souplesse ont fait lui un langage universel. En plus UML est essentiellement un support de communication, qui facilite la représentation et la compréhension de solution objet. Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation des solutions. L'aspect de sa notation, limite l'ambigüité et les incompréhensions.

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:

La vue statiques, permettant de représenter le système physiquement :

Diagrammes de classes, diagrammes d'objets, diagrammes de cas d'utilisation, diagrammes de composants, diagrammes de déploiement.

La vue dynamiques, montrant le fonctionnement du système :

Diagrammes de collaboration, diagrammes de séquence, diagrammes d'états-transitions, diagrammes d'activités.

3.2.2 Processus Unifié

L'UML propose des diagrammes pour décrire les différents aspects d'application mais ne précise pas la séquence d'étape à suivre ou la démarche à suivre pour la réalisation de ces diagrammes. Un processus de développement est alors nécessaire.

Un processus unifié est un processus de développement logiciel construit sur la notation UML. Il est itératif et incrémental, centré sur l'architecture, conduit par les cas d'utilisation et piloté par les risques.

La gestion d'un tel processus est organisée par quatre phases : pré étude, élaboration, construction et transition. Ses activités de développement sont la capture des besoins, l'analyse et la conception, l'implémentation, le test et le déploiement. [11]

3.2.3 StarUml

StarUML est un logiciel de modélisation UML open source qui peut remplacer dans bien des situations des logiciels commerciaux et coûteux comme Rational Rose ou Together . Étant simple d'utilisation, nécessitant peu de ressources système, supportant UML 2, ce logiciel constitue une excellente option pour une familiarisation à la modélisation.

3.3 Les diagrammes du système

3.3.1 Diagrammes de cas d'utilisation :

Chaque usage que les acteurs font du système est représenté par un cas d'utilisation.
Chaque cas d'utilisation représente une fonctionnalité qui leur est offerte afin de produire le résultat attendu.
Ainsi, « le diagramme de cas d'utilisation décrit l'interaction entre le système et l'acteur en déterminant les besoins de l'utilisateur et tout ce que doit faire le système pour l'acteur ».

A. Identification des acteurs :

Un acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié. [12]
On peut regrouper les acteurs de notre système dans les catégories suivantes:

- ***Secrétaire*** *:* Possède des droits sur la gestion des rendez-vous, des patients,des medecins.
- ***Médecin :*** Gestion des consultations
- ***Patient :*** Créer son compte,Demande des rendez-vous.

La figure ci-dessous représente le diagramme de cas d'utilisation général de notre système :

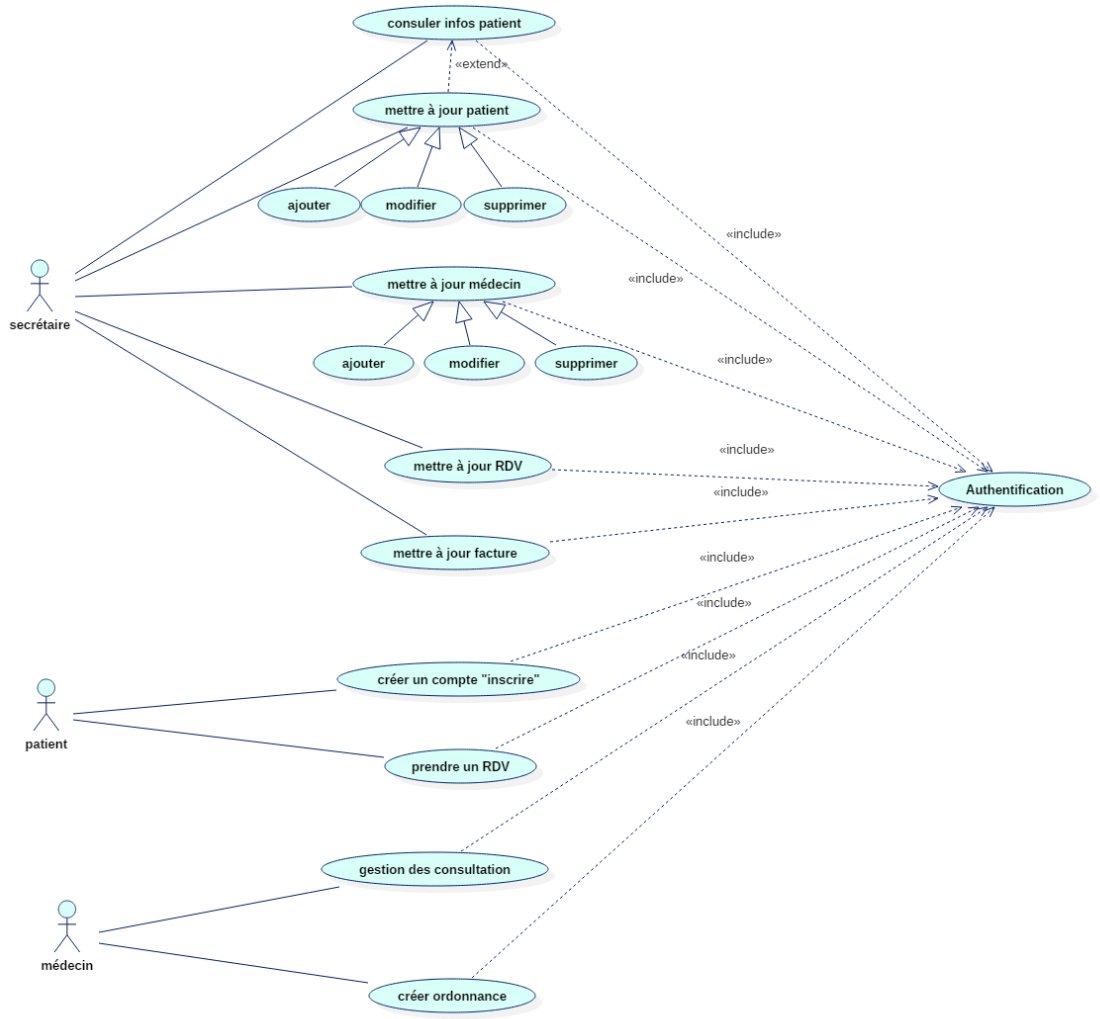

Figure 3.1 : Diagramme de cas d'utilisation

B. Identification des cas d'utilisation :

Selon le diagramme de cas d'utilisation de notre système on distingue les cas d'utilisation suivants :

Authentification : l'application vérifie que l'utilisateur est bien ce qu'il prétend être et lui donne ensuite l'autorisation d'accès.

Mettre à jour patient : pouvoir ajouter, modifier, supprimer un patient.

Mettre à jour médecin : pouvoir ajouter, modifier, supprimer un médecin.

Consulter informations patient : la secrétaire peut chercher un patient .

Mettre à jour facture: pouvoir ajouter, modifier, supprimer une facture.

Créer un compte « inscrire » : pour que le patient peut accéder à l'application ,il faut qu'il doit avoir un compte.

Prendre un Rendez-vous : le patient prend un rendez-vous en choisissant une date et un service.

Gestion des consultations : le médecin fait une consultation et créer une ordonnace.

C. Les relations entre cas d'utililisation :

Dans notre diagramme de cas d'utilisation nous avons utilisé les relations d'inclusion et d'extension qui servent à enrichir des cas d'utilisation par d'autres cas d'utilisation.

L'inclusion « include » : Cela implique obligatoirement l'inclusion d'un cas d'utilisation dans un autre comme ici « prendre un rendez-vous » fait obligatoirement appel à « S'authentifier »

L'extension « extended » : Cela permet éventuellement l'extension d'un cas d'utilisation par un autre comme ici « mettre à jour patient » peut étendre « Consulter informations patient ».

3.3.2 Diagramme de séquence

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation UML.
Dans ce qui suit, nous présentons quelques diagrammes de séquence de notre système.

Diagramme de séquence pour le cas « Authentification »

Avant d'entrer au menu du projet et faire l'ensemble des autres scénarios l'utilisateur doit se connecter en utilisant son login plus son mot de passe.

Le système vérifie la validité des donnés, il affiche la page d'accueil si les données sont correctes sinon il affiche un message d'erreur. Le diagramme qui suit présente l'enchainement de la phase d'authentification.

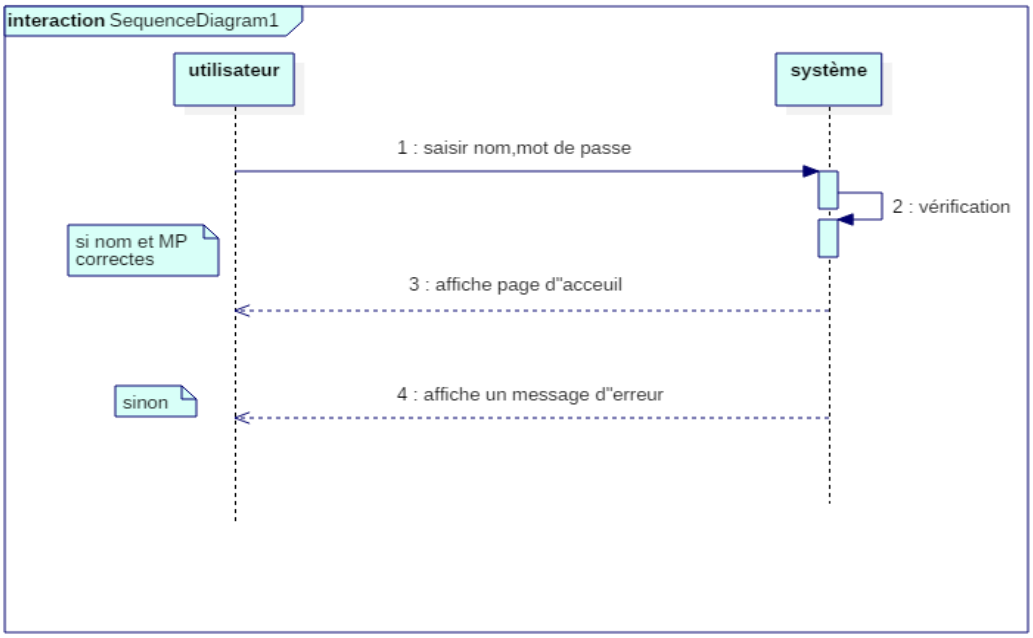

Figure 3.2 : Diagramme de séquence du système : Authentification

Diagramme de séquence pour le cas « création d'un compte »

Cette étape est obligatoire pour que le patient peut se connecter au site. . Une fois Le patient demande la connexion, le système va afficher un formulaire d'inscription. Le patient doit remplir les champs obligatoires et le système vérifie la validité des données avant de valider.

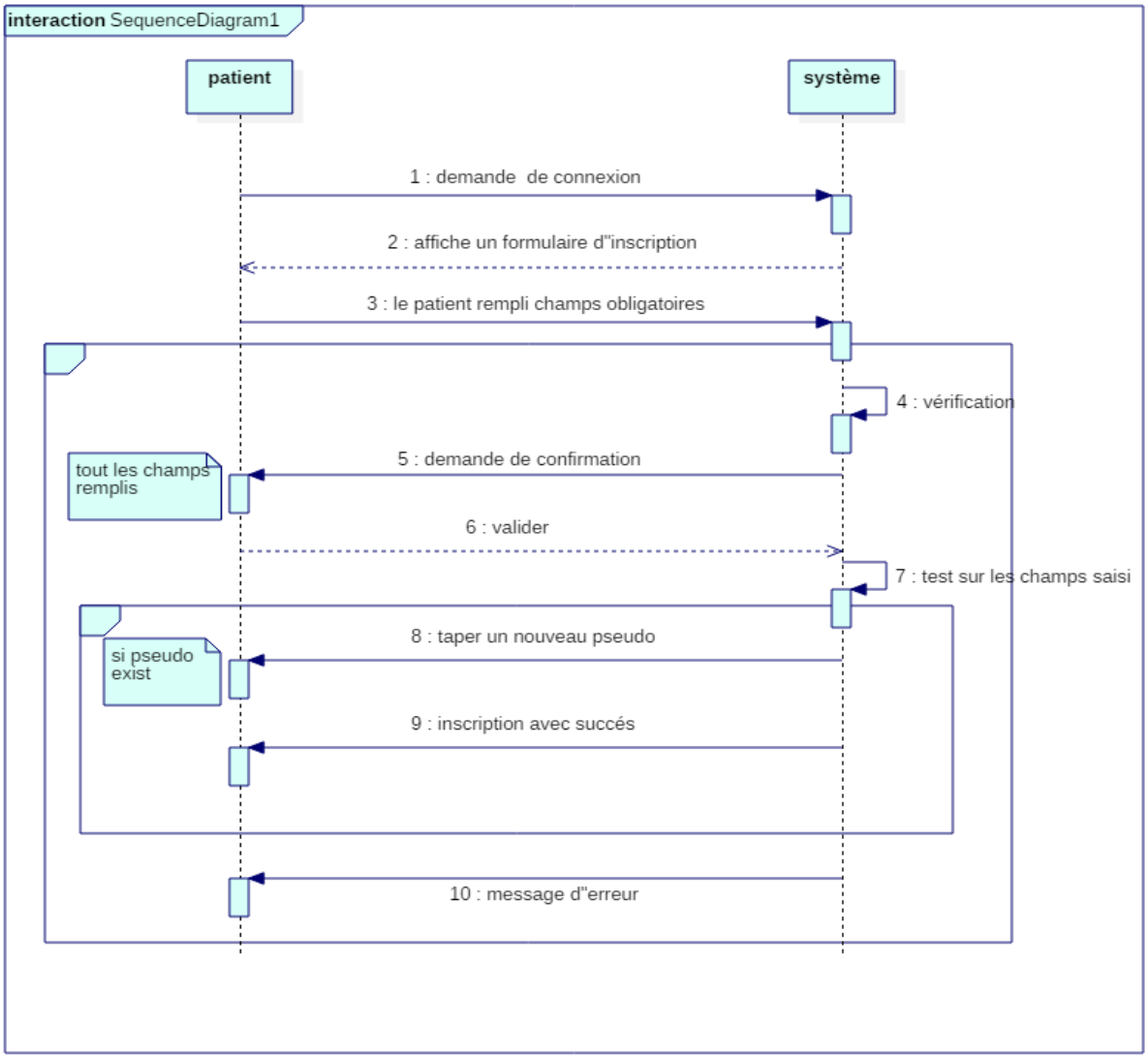

Figure 3.3 : diagramme de séquences du cas d'utilisation « créer compte ».

Diagramme de séquence pour le cas « prendre rendez-vous»

Après que le patient s'authentifie , le système va donner la main pour prendre un rendez-vous . Alors le patient doit remplir le formulaire de rendez-vous en choisisant le service et la date .

Les informations seront enregistrés une fois qu'elles seront vérifié par le système.

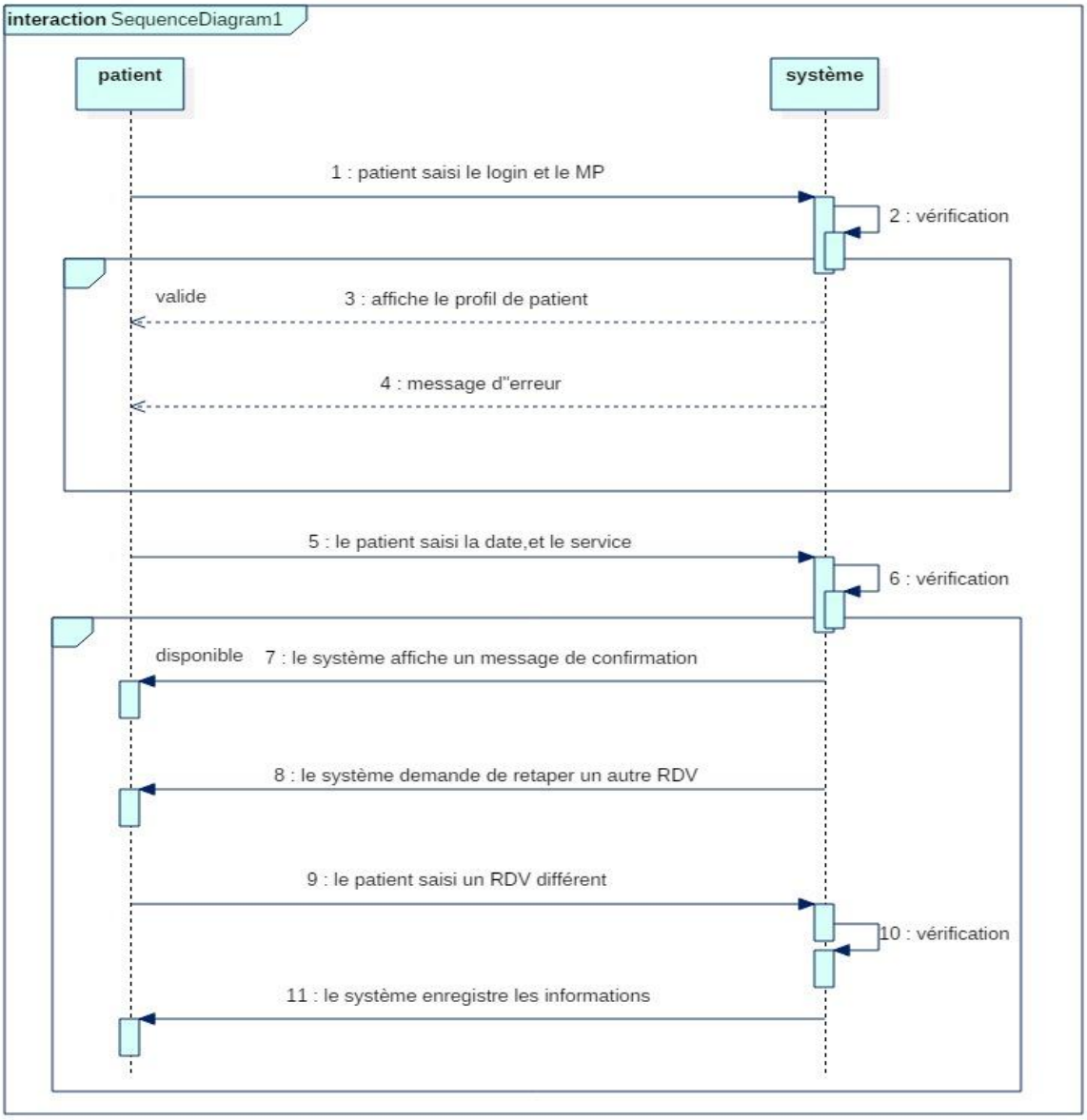

Figure 3.4 : diagramme de séquences du cas d'utilisation « prendre rendez-vous ».

.

3.3.3 Diagramme de classes

Le diagramme de classes représente les classes constituant le système et les associations entre elles.

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classe et de relations entre ces classes.

De même qu'une classe décrit un ensemble d'objets, une association décrit un ensemble de liens ; les objets sont des instances de classes et les liens sont des instances de relations.

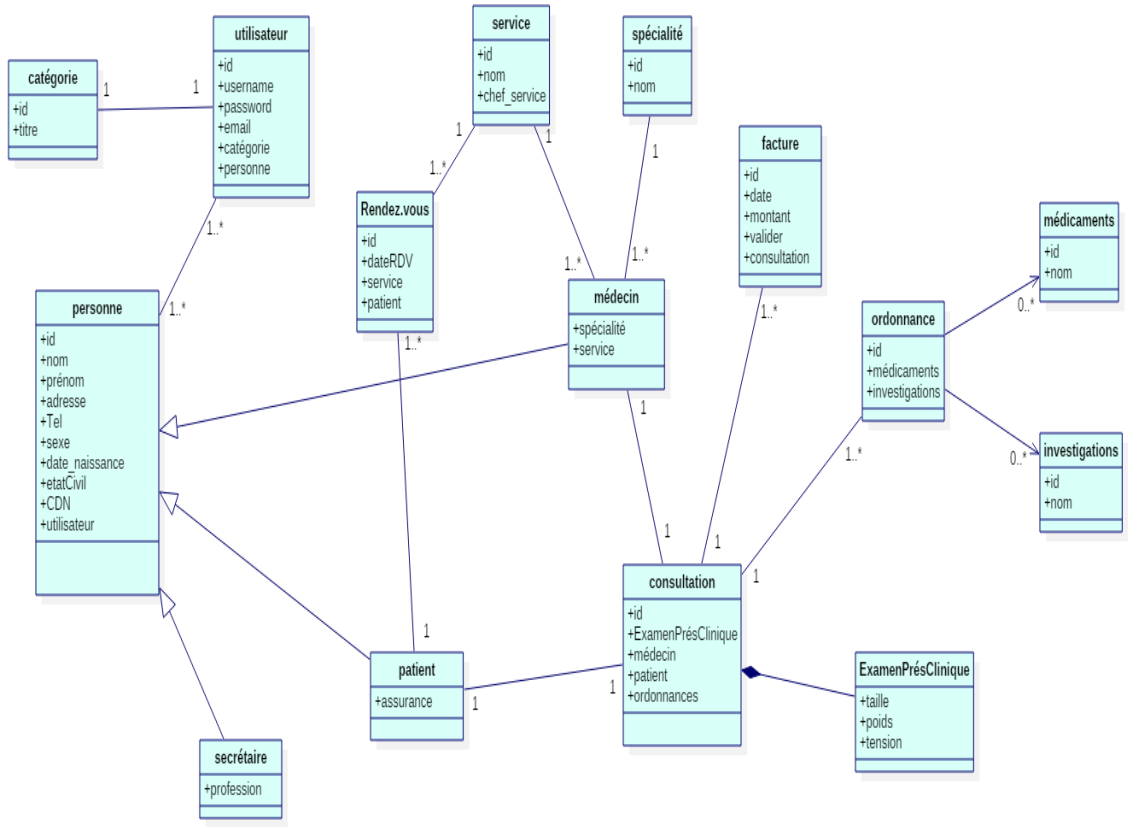

Figure 5.5 : diagramme de classe

D'après l'étude du système existant et des différents diagrammes de cas d'utilisation, nous avons pu dégager les principales classes illustrées dans la figure ci-dessus pour avoir une vue plus claire du système étudié.
A partir de ce diagramme, on dégage les entités de la base de données ainsi que les java beans correspondant dans l'application à développer.
Ce diagramme de classe ne contient pas des méthodes.

3.3.4 Le modèle relationnel

Pour passer du diagramme de classe vers le modèle logique il y a des règles qu'il faut les respectés.

Règle 1 : présence de la cardinalité (? .. 1) d'un coté de l'association :

- Chaque classe se transforme en une table.
- Chaque attribut de classe se transforme en un champ de la table.
- L'identifiant de la classe qui est associé à la cardinalité (? ..1) de l'autre classe.

Règle 2 : présence de (? .. N) des deux cotés de l'association :
Chaque classe se transforme en une table.

- Chaque attribut de classe se transforme en un champ de table.
- L'association se transforme en une table. Cette table a comme champs l'identifiant de chacune des deux classes, plus d'éventuels autres attributs.

Règle 3 : présence d'une généralisation :

Méthode 1:

- Création d'une table avec tous les attributs des classes.
- L'ajout d'un attribut pour distinguer les types des objets.

Méthode 2 :

- Création d'une table pour chaque sous type, chaque table se compose des attributs Génériques et d'attributs spécifiques.

Méthode 3 :

- Création d'une table par classe et des associations.

La figure 3.6 représente le modèle relationnel de notre système.

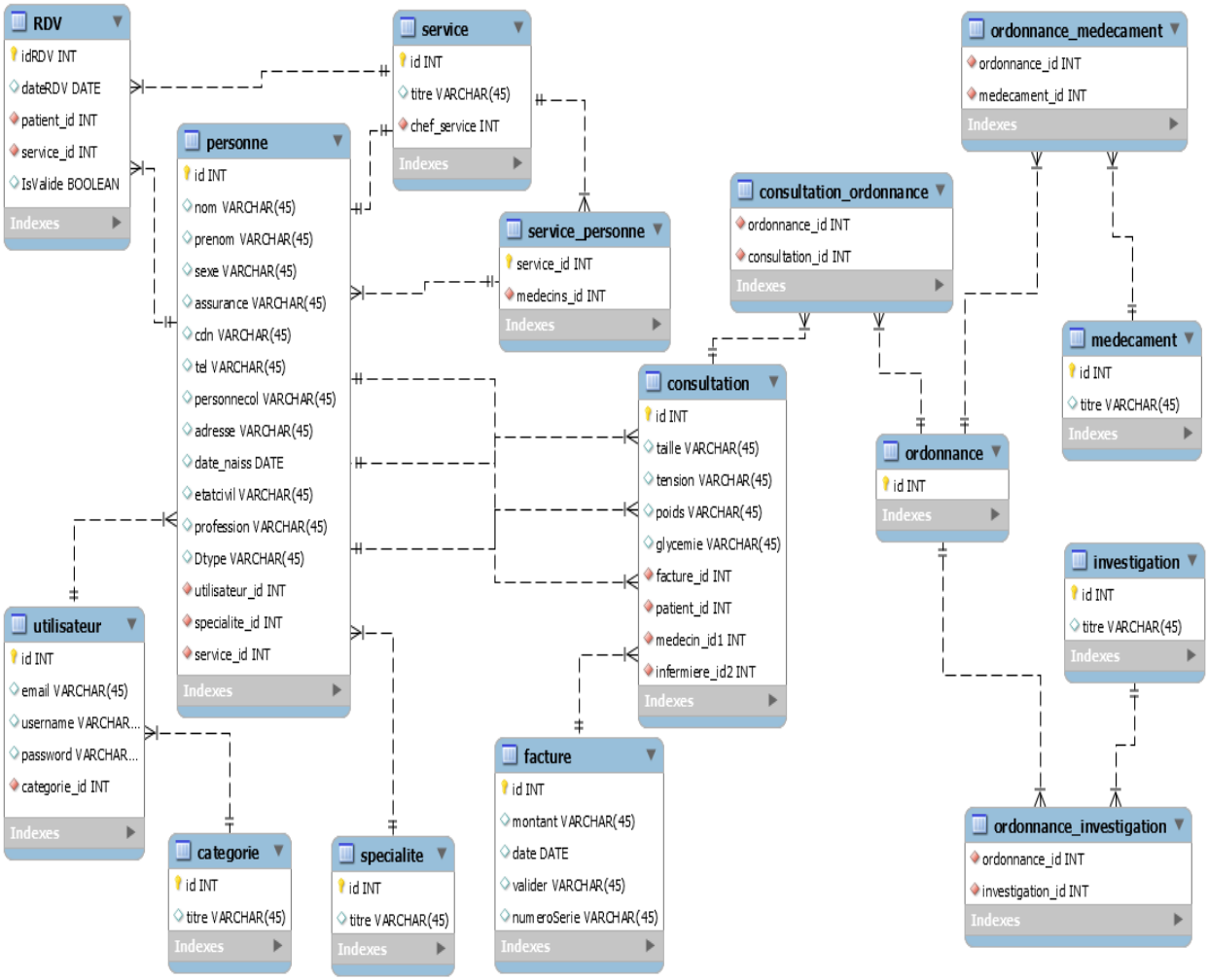

Figure 3.6 : Modèle logique de données relationnelles.

3.4 Conclusion

Comme nous pouvons le constater, l'activité de la conception a facilité la compréhension de notre système, qui ébauche vers l'activité d'implémentation.

4. REALISATION DU SYSTEME

4.1 Introduction

Après avoir achevé l'étape de conception de l'application, on va entamer dans ce Chapitre la partie réalisation et implémentation dans laquelle on s'assure que le Système est prêt pour être exploité par les utilisateurs finaux.
Nous présentons l'architecture sur laquelle nous avons développé notre application, les différents outils et techniques utilisés ainsi quelques interfaces.

4.2 Architecture technique

L'architecture technique (également nommée architecture physique) décrit le matériel supportant l'application.
Le choix s'est porté sur une architecture trois tieres (3/3), comportant d'un côté le serveur web de traitement des requêtes où est stockée notre application, d'un serveur de base de données, et d'un poste de travail munis d'un browser permettant l'accès à notre application et donc à la formulation des requêtes.
L'architecture trois tiers est un modèle logique d'architecture applicative qui vise à séparer très nettement des couches logicielles au sein d'une même application et à modéliser et présenter cette application comme un empilement de trois couches, étages, niveaux ou strates dont le rôle est clairement défini :
La présentation des données : correspondant à l'affichage, la restitution sur le poste de travail et le dialogue avec l'utilisateur.
Le traitement métier des données : correspondant à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative.
L'accès aux données persistantes: correspondant aux données qui sont destinées à être conservées sur la durée, voire de manière définitive.

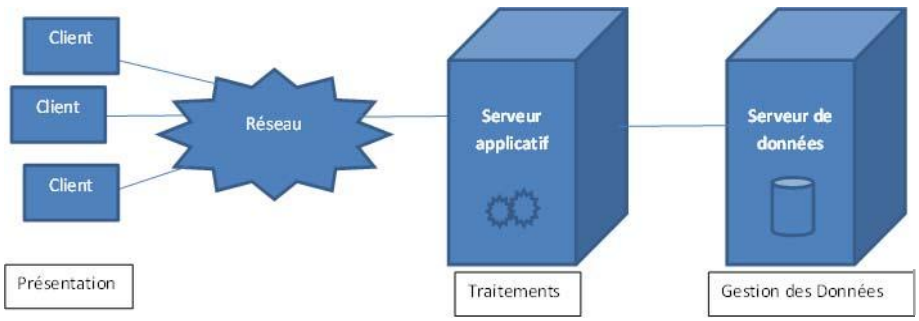

Figure 4.1 : Architecture 3-tieres

4.3 Architecture MVC :

Le modèle de conception que nous allons choisir et qui satisfait notre besoin est le MVC (Modèle-Vue-Contrôleur).

MVC est une méthode de conception qui organise l'interface homme-machine (IHM) d'une application logicielle. Ce paradigme divise l'IHM en (voir figure 16)

- **un Modèle :** cette partie gère les données du site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc les requêtes SQL.
- **une Vue :** cette partie se concentre sur l'affichage, Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit affiche.
- **un Contrôleur :** cette partie gère la logique du code qui prend des décisions.

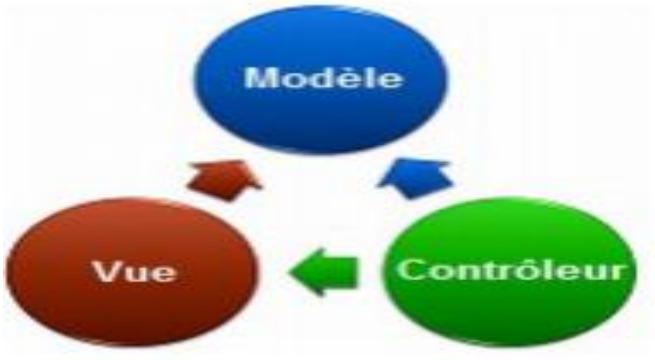

Figure 4.2 : Modèle-Vue-Contrôleur

4.4 Choix des outils et langages de développement

Pour la réalisation de ce projet nous avons choisi de travailler avec :

NetBeans : C'est un environnement de développement intégré(EDI), placé en open source par Sun en juin 2000 sous licence CDDL (Common Développent and Distribution License) et GPLv2. En plus de Java, NetBeans permet la prise en charge native de divers langages tels le C, le C++, le JavaScript, le XML, le PHP et le HTML, .

Il offre toutes les facilités d'un IDE moderne (éditeur en couleurs, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web). Compilé en Java. NetBeans constitue par ailleurs une plate forme qui permet le développement d'applications spécifiques.[wiki]

***Heidi MySQL* :** C'est un outil d'administration de base de données possédant un éditeur SQL et un constructeur de requête. Il a été développé et optimisé pour être utilisé avec le SGBD relationnel MySQL disponible commercialement ou gratuitement.
Il est capable de se connecter à des bases MySQL, SQL Server. [2]

Serveur d'applications glassfish : notre choix est porté sur glassfish car il fournit toute sorte de services standardisé: conteneur d'EJB, gestion de transactions, gestion de la sécurité, gestion du déploiement…
Glassfish est le nom du serveur d'application Open Source java EE 5 et désormais java EE 7 avec la version 1.4 qui sert de socle au produit Oracle Glassfish.
De plus, glassfish permet de se connecter à la plupart des standards du marché (ORACLE, MySQL, HeidiSQL…).

HTML & CSS :
HTML : est un langage dit de « marquage » (de « structuration » ou de « balisage ») dont le rôle est de formaliser l'écriture d'un document avec des balises de formatage. Les balises permettent d'indiquer la façon dont doit être présenté le document et les liens qu'il établit avec d'autres documents. [1]
CSS: feuille de style c'est un langage qui permet de gérer l'apparence de la page web (agencement, positionnement, décoration, couleur, taille du texte...).

JQUERY : est une bibliothèque JavaScript libre qui porte sur l'interaction entre JavaScript (comprenant Ajax) et HTML, et a pour but de simplifier des commandes communes de JavaScript. [1]

La technologie JSP : c'est une technique basée sur Java qui permet aux développeurs de créer dynamiquement du code HTML, XML ou tout autre type de page web. Cette technique permet au code Java et à certaines actions prédéfinies d'être ajoutés dans un

contenu statique. Depuis la version 2.0 des spécifications, la syntaxe JSP est complètement conforme au standard XML.

Le langage JPQL : est un langage de requête dont la grammaire est définie par la spécification JPA. Il est très proche du langage SQL dont il s'inspire fortement mais offre une approche objet. On trouve également le nom d'EJBQL dans la littérature Java, il s'agit du nom donné à ce langage dans la norme EJB2.
on l'utilise pour exécuter des requêtes directement sur les entités bean.
Un exemple d'une requête JPQL :

```
@PersistenceContext em;
Query query = em.createQuery ("select object (o) from Patient as o ");
```

Entreprise JavaBeans EJB : Notre choix s'est porté sur les EJB 3.1 car ils intègrent la JPA (Java Persistance API) qui est utilisée au niveau de la couche de persistance pour créer une séparation entre l'application et la base de données. Ils sont parfaitement adaptés pour être intégrés dans une architecture trois tiers ou plus.
Les types de composants EJB que nous avons utilisés sont :

Les Sessions beans : qui sont citées au dessous :

- CategorieFacade qui implémente l'interface 'categorieFacadeLocal'.
- ConsultationFacade implémente l'interface 'consultationFacadeLocal'.
- FactureFacade implémente l'interface FactureFacadeLocal.
- InfermiereFacade implémente l'interface InfermiereFacadeLocal.
- InvestigationFacade implémente l'interface InvestigationFacadeLocal.
- MedecinFacade implémente l'interface MedecinFacadeLocal.
- MedicamentFacade implémente l'interface MedicamentFacadeLocal.
- OrdonnanceFacade implémente l'interface OrdonnanceFacadeLocal.
- PatientFacade implémente l'interface PatientFacadeLocal.
- PersonneFacade implémente l'interface PersonneFacadeLocal.
- RDVFacade implémente l'interface RDVFacadeLocal.
- SecretaireFacade implémente l'interface SecretaireFacadeLocal.
- ServiceFacade implémente l'interface ServiceFacadeLocal.

- SpecialiteFacade implémente l'interface SpecialiteFacadeLocal.
- UtilisateurFacade implémente l'interface UtilisateurFacadeLocal.

Pour faciliter l'implémentation des méthodes nous avons utilisé le paterne de conception « **Façade** » pour le but de cacher une conception et une interface complexe difficile à comprendre (cette complexité étant apparue « naturellement » avec l'évolution du sous-système en question).

Façade permet d'encapsuler la complexité des interactions entre les objets métier.

Une façade peut être utilisée pour :[2]

- rendre une bibliothèque des méthodes plus facile à utiliser, comprendre et tester;
- rendre une bibliothèque des méthodes plus lisible;
- réduire les dépendances entre les clients de la bibliothèque et le fonctionnement interne de celle-ci, ainsi on gagne en flexibilité pour les évolutions futures du système.

Chaque interface contient des méthodes qui sont implémentées dans la session.

Un exemple d'interface ***PatientFacadeLocal*** :

```
@Local
 public interface PatientFacadeLocal {
  void create (Patient patient);
  void edit (Patient patient);
  void remove (Patient patient);
  Patient find (Object id);
  List<Patient> findAll();
  List<Patient> findRange(int [] range);
  int count();
 }
```

Un exemple d'implémentation de **la méthode create (ajouter un patient) :**

```
public void create(Patient patient) {
  EntityManager em = null;
   try {
      em = getEntityManager();
      em.getTransaction().begin();
      Utilisateur utilisateur = patient.getUtilisateur();
      if (utilisateur != null) {
         utilisateur = em.getReference(utilisateur.getClass(), utilisateur.getId());
         patient.setUtilisateur(utilisateur);
      }
      em.persist(patient);
      if (utilisateur != null) {
         entity.Personne oldPersonneOfUtilisateur = utilisateur.getPersonne();
         if (oldPersonneOfUtilisateur != null) {
            oldPersonneOfUtilisateur.setUtilisateur(null);
            oldPersonneOfUtilisateur = em.merge(oldPersonneOfUtilisateur);
         }
         utilisateur.setPersonne(patient);
         utilisateur = em.merge(utilisateur);
      }
      em.getTransaction().commit();
   } finally {
      if (em != null) {
         em.close(); }   }}
```

Un exemple d'implémentation de la méthode **findPatientEntities (chercher un patient) :**

```
public Patient findPatient(Long id) {
    EntityManager em = getEntityManager();
    try {
       return em.find(Patient.class, id);
    } finally {
       em.close(); }
```

Les Entités bean :

Pour créer nos entités beans, nous avons d'abord créé le schéma des classes puis à développer les entités beans. Nous avons utilisé donc le diagramme des classes, puis nous avons généré automatiquement les entités beans par la suite.

Les entities de notre application sont :

- Categorie.
- Consultation.
- ExamenPreClinique.
- Facture.
- Infermiere.
- InfoCivil.
- InfoPersonnel.
- Investigation.
- Medecin.
- Medicament.
- Ordonnance.
- Patient.
- Personne.
- RDV.
- Secretaire.
- Service.
- Specialite.
- Utilisateur.

Un exemple **d'entité Patient** :

```
@Entity
 public class Patient extends Personne implements Serializable {
   @Column(unique = true)
   @Basic
   private String numero_patient;
   public String getNumero_patient() {
      return this.numero_patient;
   }
   public void setNumero_patient(String numero_patient) {
      this.numero_patient = numero_patient;
```

4.5 Développement de l'application

4.5.1 Principales interfaces graphiques :

La conception des interfaces de l'application est une étape très importante puisque toutes les interactions avec le cœur de l'application passent à travers ces interfaces, on doit alors guider l'utilisateur avec les messages d'erreurs et de notification si besoin, ainsi présenter un système complet.

Dans cette partie, nous allons présenter quelques interfaces de notre l'application.

A. Interface Page d'accueil :

Notre application a 2 pages d'accueil différentes selon l'utilisateur :

la première pour le patient et la deuxième commune entre la secrétaire et le médecin.

Page d'accueil patient :

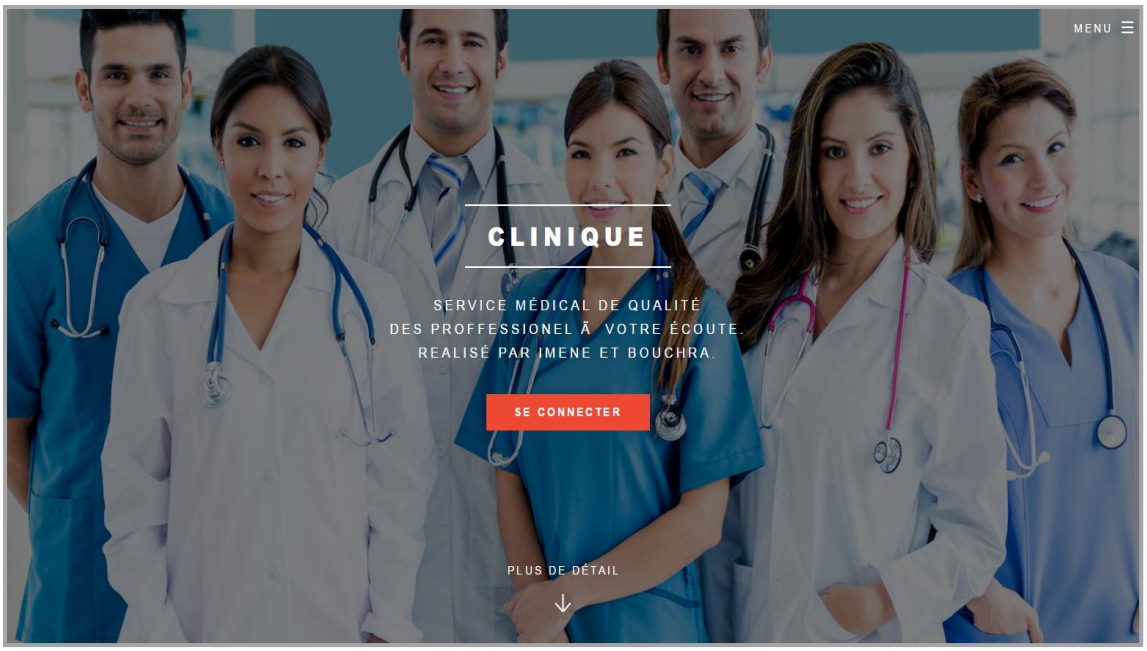

Figure 4.3 : Page d'accueil patient

Page d'accueil AdminPage

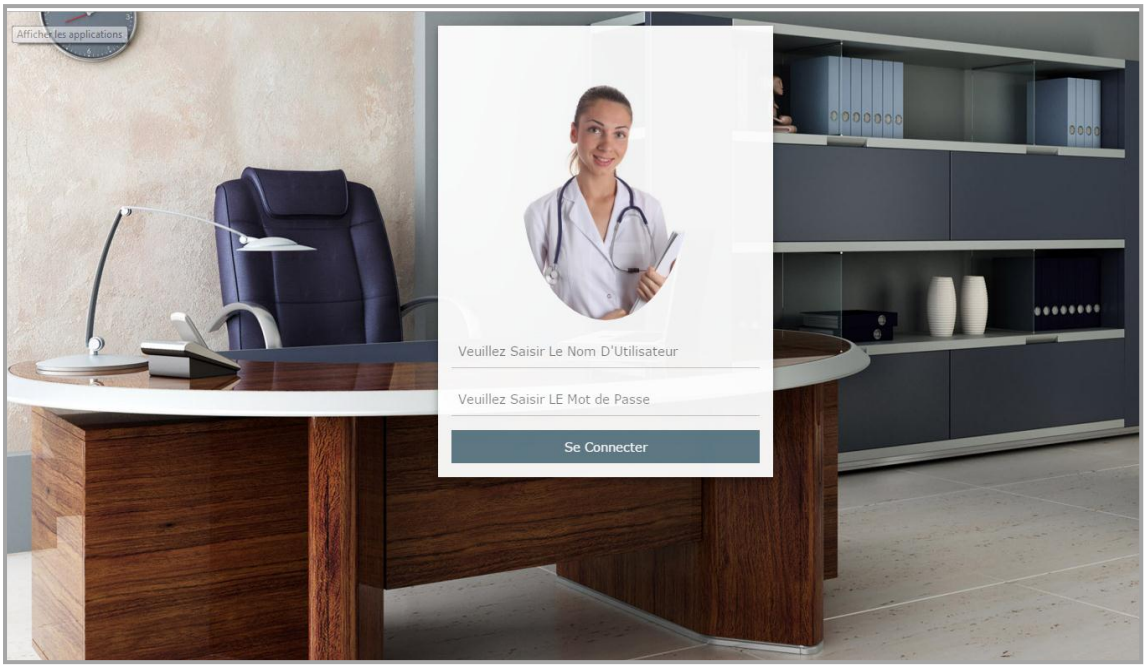

Figure 4.4 : Page d'accueil AdminPage

B. Interface d'inscription des patients :

Une fois le patient clique sure s'enregistrer le formulaire d'inscription s'affiche à l'écran.

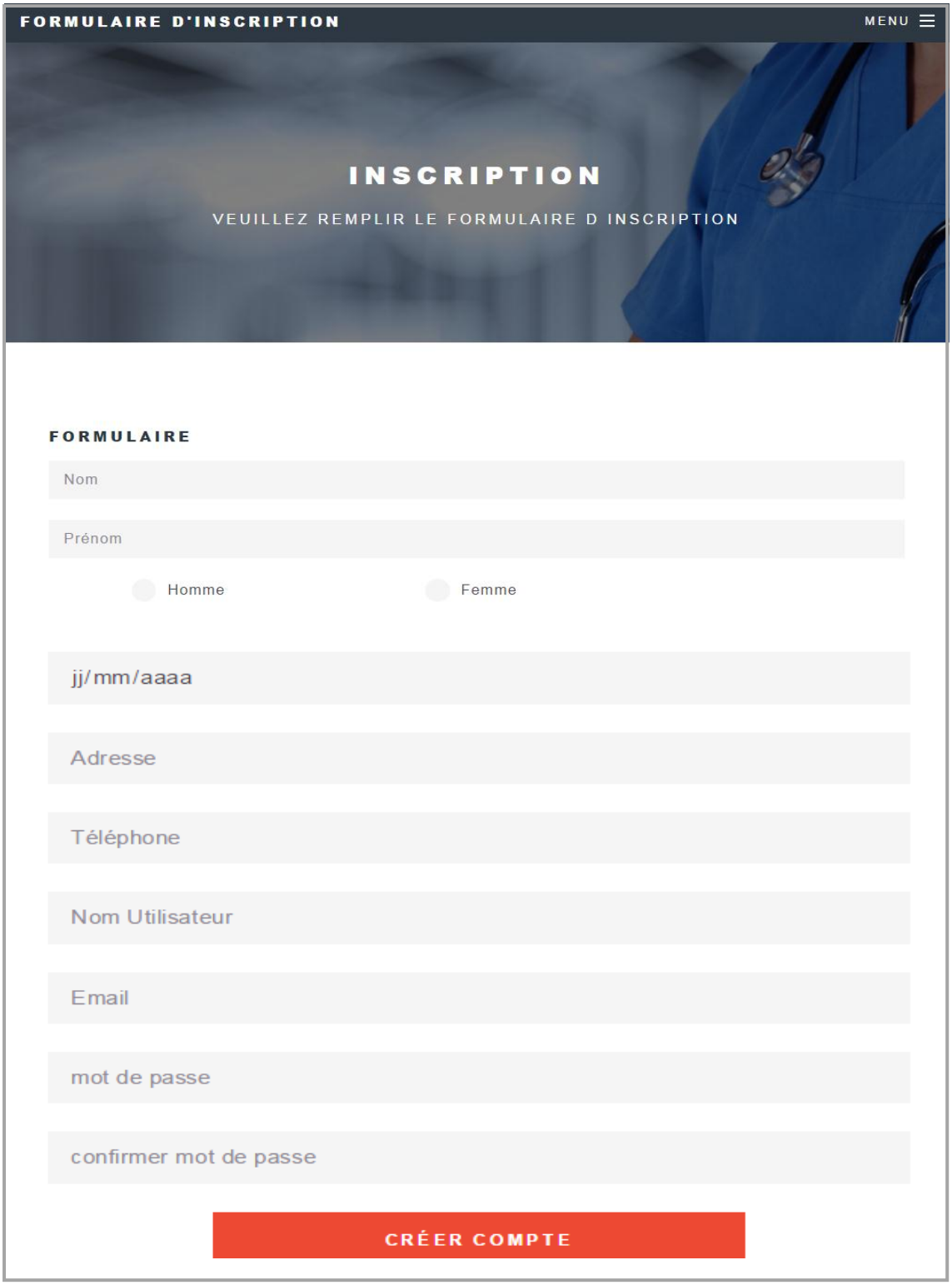

Figure 4.5 : interface inscription patient

C. Interface ajouter médecin :

A gauche de la figure suivante apparait le menu secrétaire .

Parmi ses taches principales c'est ajouter un nouveau médecin au clinique.

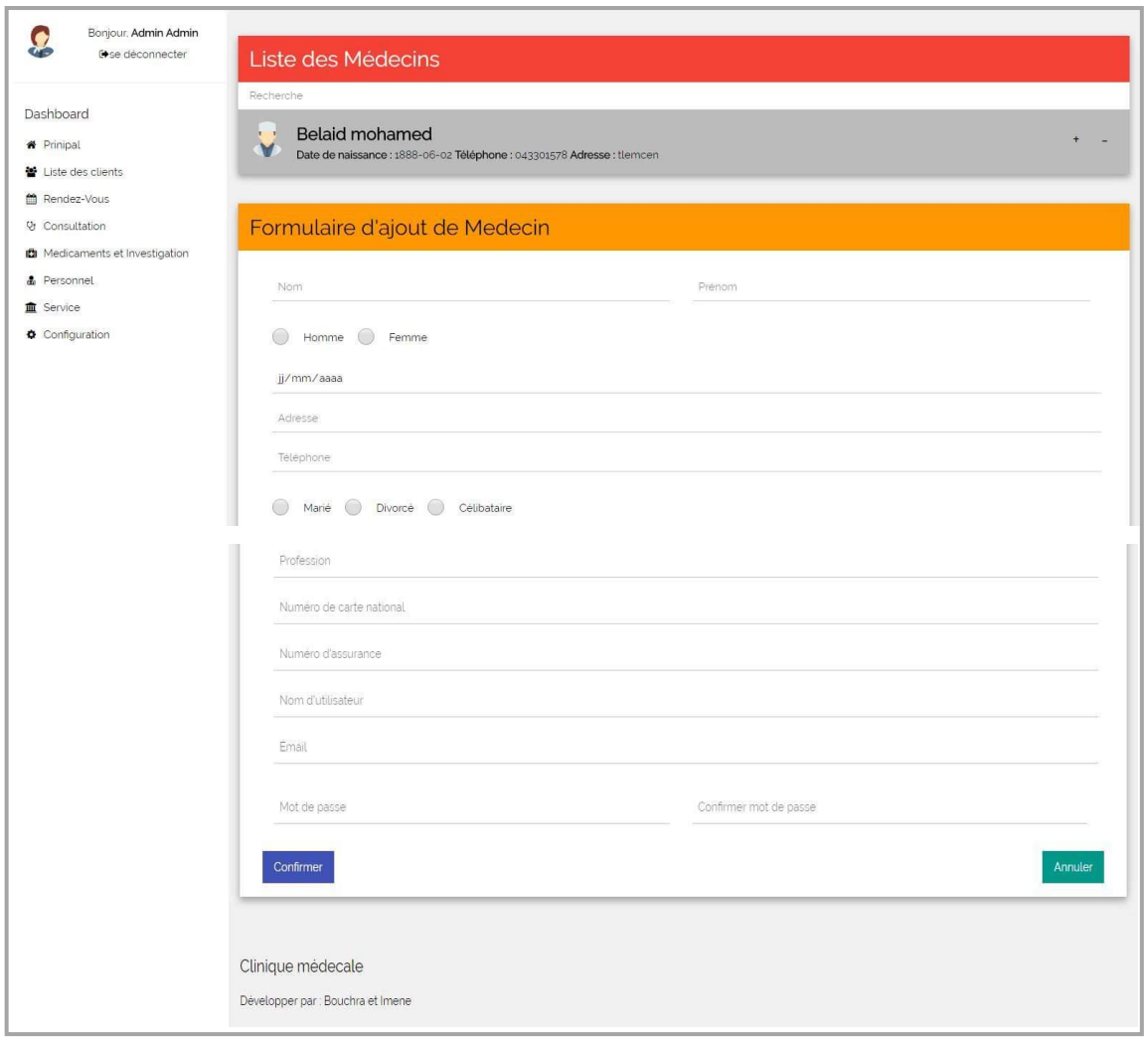

Figure 4.6 : interface ajouter médecin

D. Interface de consultation :

Une fois le médecin s'authentifie l'interface suivant est apparait.

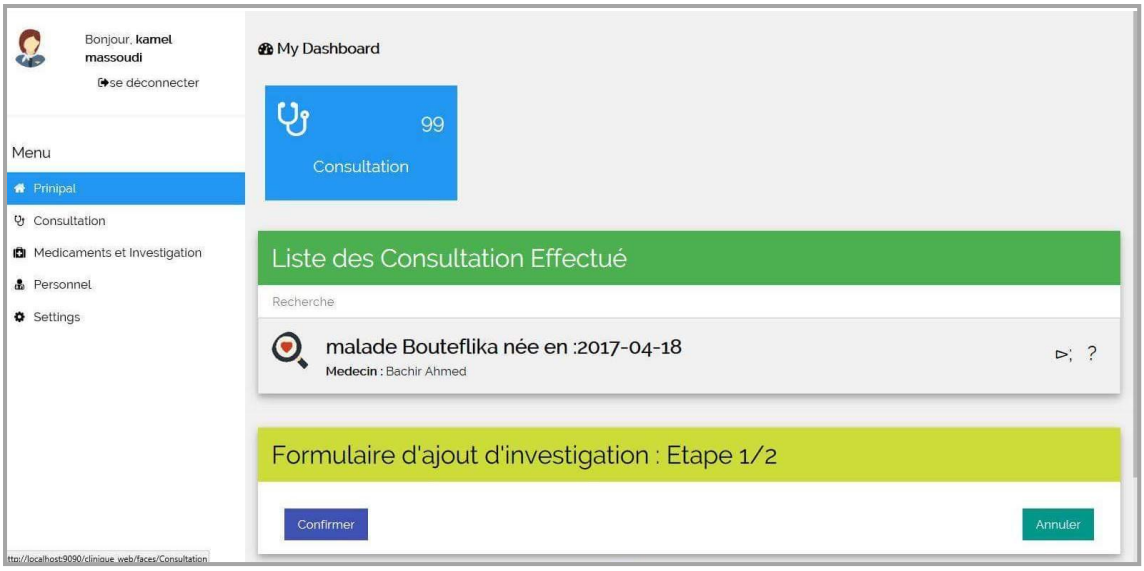

Figure 4.7 : interface consultation

E. Interface ajouter Service :

La secrétaire peut aussi ajouter un service et voilà l'interface correspondant dans la figure ci-dessous :

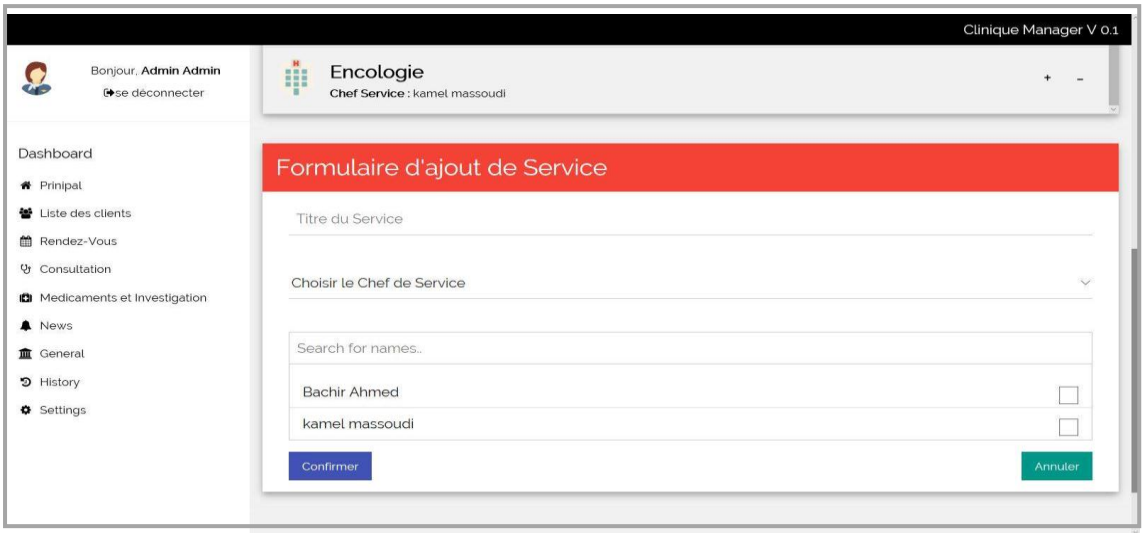

Figure 4.8 : interface ajouter service

F. Interface gstion rendez-vous :

la gestion des rendez-vous est parmi les plus importantes taches de la secrétaire.
Elle peut valider un rendez-vous une fois le patient présentera dans la clinique sinon elle peut aussi l'annuler.

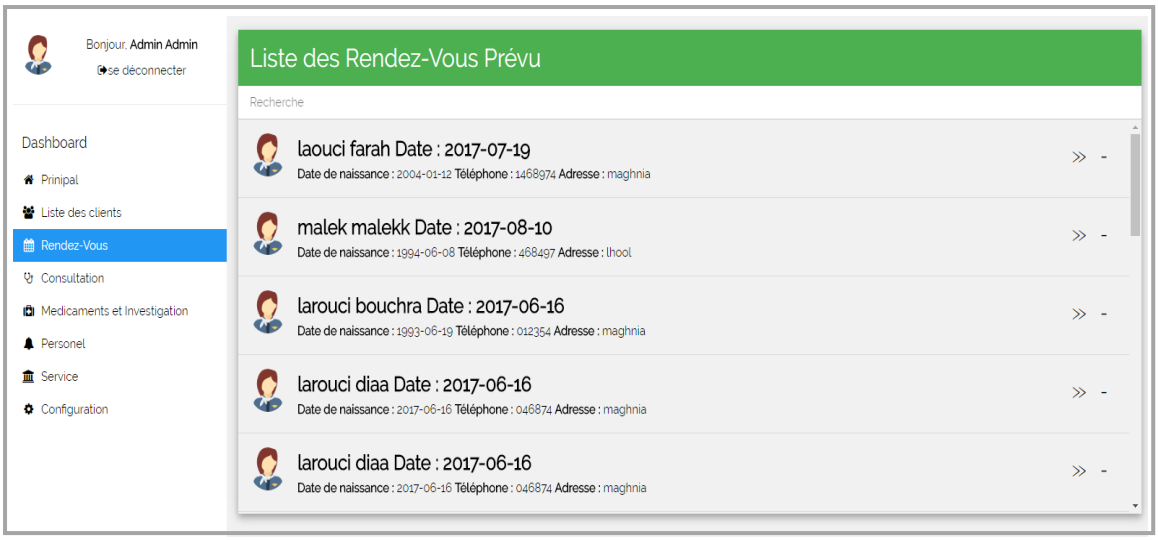

Figure 4.9 : interface gestion des rendez-vous

CONCLUSION GENERALE

Conclusion Générale

L'objectif de notre projet est la réalisation d'une application médicale qui aide à la gestion et l'informatisation des patients dans une clinique, et à encourager les gens à utiliser la technologie dans leur travail pour le faciliter et le rendre moins compliqués et plus sécurisé. Le développement de notre application web dans le domaine médicale nous a permis de découvrir un nombre d'outils d'informatique, ainsi que l'importance de la sécurité des données.

Tous d'abord nous avons commencé par la conception, en utilisant le formalisme UML puisque notre application devra rester ouverte pour les améliorations futures.

Pour les données, nous avons eu recours à la base de données Heidi SQL pour sa capacité de stockage adéquate au projet développé ainsi que pour son aptitude à stocker les images sous bonne qualité.

Nous avons acqui de nouvelles connaissances en EJB et de découvrir des nouveaux design patterns, notamment MVC qui permet une séparation et une distribution des tâches entre développeurs, et rend l'application plus flexible. Aussi le pattern Façade qui cache la conception et la complexité de code source et le rend plus lisible.

Ceci nous permis d'avoir une bonne expérience et une amélioration de nos connaissances concernant le domaine de la programmation Orienté Objet.

Enfin nous pensons que ce système peut être amélioré pour le rendre aussi convenable dans l'avenir.

BIBLIOGRAPHIE

[1] :Mammeri – UPS, CORBA :principes et mécanismes Développement d'applications distribuées

[2] : Michel RIVEILL, David EMSELLEM, Antoine BEUGNARD,2007,Intergiciel et Construction d'Applications Réparties.

[3] : JavaSoft : http://java.sun.com/j2ee/

[4] :Jean-Philippe ,FORESTIER, Comparaison des architectures J2EE et .NET, OSYX 2003

[5] :Mr BENMARZOUKA HABIB REDHA,MR MALIKI Fouad,Modelisation-et-implementation-dune-application-web-e-commerce-dans-la-plate-forme-J2EE mémoire de fin d'études,2007.

[6] : Pierre-Yves Gibello, Java EE Java Enterprise Edition, Septembre2011.

[7] : Jean-Michel DOUDOUX, Copyright (C) 1999-2016,Développons en Java.

[8] : Michel Buffa,24/11/2011, Cours EJB/J2EE.

[9] : Michel Buffa, UNSA 2002, EJB : les fondamentaux.

[10] : Didier DONSEZ, Université Joseph Fourier (Grenoble 1) IMA – LSR/ADELE ,Les Enterprise Java Beans.

[11] : Pascal Roques, Franck Vallée « UML2 en action, De l'analyse des besoins à la conception», Edition : EYROLLES 2007.

[12] : Pascal Roques, UML2 par la pratique, Etudes de cas», 2008.

[13] : Laboratoire SUPINFO, EJB 3 Des concepts à l'écriture du code, 2006.

[14] : Laurent DEBRAUWER, Design Patterns en Java, 2013.

LISTE DES FIGURES

RESUME

Notre projet consiste a développer une application distribuée, basée sur le principe du client/serveur, et plus précisement l'architecture 3-tières
nous choisissons d'utiliser Le modèle MVC(modèle vue controleur) est une méthode de conception utilisée pour rester dans le principe de l'autonomie, les EJB (entreprise java beans) qui est un composant conçu pour créer la connexion entre la base de données et le serveur aussi pour assurer la réutilisabilité.
Pour la réalisation de cette application, nous avons choisi pour la conception le langage de modélisation UML et plus précisément le processus unifié UP (Unified Proces), MYSQL comme un SGBD pour construire et gérer notre base de données et ensuite Java avec IDE NetBeans pour la programmation et le développement de notre système.

Mots-clés : client/serveur, architecture 3-tièrs, MVC, EJB,MYSQL,Java

ABSTRACT

Our project consist a distributed application, based on the client / server, and more precisely the 3-tires architecture.
We choose to use the MVC (model view controller) design model used to stay in the principle of autonomy, EJBs (enterprise java beans) which is a constituent aims to create the connection between the database and the server also to ensure reusability.
To implement this application, we chose the UML modeling language and more precisely the unified process UP (Unified Proces). MSQL as a DBMS to build and manage our database than Java with IDE NetBeans for programming and the development of our system.

Keywords: client / server, 3-tiers architecture, MVC, EJB, MYSQL, Java

ملخص

يتمثل مشروعنا في انجاز تطبيق موزع، يستند على مبدأ client/serveur، وبصورة أدق بنية 3tièrs.
استخدمنا نموذج التصميم MVC (نموذج نظرة مراقب) و هو عبارة عن طريقة للتصميم من اجل الحفاظ على مبدأ الحكم الذاتيEJBs (مؤسسة جافا بين) لإنشاء الاتصال بين قاعدة البيانات و serveurوضمان إعادة الاستخدام.
لتنفيذ هذا التطبيق، اخترنا لغة النمذجة UML وتحديدا UP (العملية الموحدة) . MSQL لإدارة قواعد البيانات الخاصة بناء، واخيرا Java IDE للبرمجة و تطوير نظامنا.

الكلمات المفتاحية : Client / server, architecture 3-tièrs, MVC, EJB, MYSQL, Java