CHAPITREIII

Modélisation et conception

1. Introduction

De nos jours, les bases de données sont des composants incontournables de serveurs Web et d'applications en ligne, qui fournissent du contenu dynamique. Ils permettent d'organiser efficacement la sauvegarde et la lecture des données d'un programme, des données secrètes ou critiques peuvent être stockées dans les bases de données, il est donc important de les protéger efficacement.

Pour lire ou stocker des informations, vous devez vous connecter au serveur de bases de données, envoyer une requête valide, lire le résultat et refermer la connexion. De nos jours, le langage le plus courant pour ce type de communication est le langage SQL (*Structured Query Language*).

Dans le cadre de notre PFE est afin d'implémenter des mécanismes de sécurité, nous proposerons de réaliser une application bancaire qui intégrera un ensemble de contrôle d'accès. Pour faire cela, et dans le cadre de ce chapitre, nous présenteront la modélisation de notre application bancaire qui intègre des contrôles d'accès que nous avons proposés. La modélisation est basée sur le langage UML (Unified Modelisation Language). L'application qui sera présentée dans le chapitre suivant est basée sur deux types d'architecture : 2-tiers pour implémenter les rôles d'Agent et Administrateur ici, ou on a utilisé un modèle à base de rôle et 3-tiers pour gérer le client avec des propositions pour la gestion de la sécurité dans ce cas.

2. Environnement du projet - outils utilisés

Dans ce projet, nous avons utilisé le langage de programmation Java sous NetBeans 6.8, en particulier, les JSP (Java Server Pages) pour les pages Web dynamiques, et aussi EasyPHP pour administrer notre base de données.

2.1. Java sous NetBeans

Java est un langage robuste qui peut être exploité pour développer un large éventail de programmes utilisant une interface utilisateur graphique, pouvant être appliqués en réseau pour se connecter à des bases de données, et offrant d'autres fonctionnalités toutes plus sophistiquées les unes que les autres.

Le ramasse-miettes intégré à Java détecte automatiquement les objets inutilisés pour libérer la mémoire qu'ils occupent. Aussi le Java intègre la gestion des exceptions pour faciliter la mise au point des programmes (détection et localisation des bugs). Concernant la sécurité, Java protège les informations sensibles de l'utilisateur et le système d'exploitation de sa machine en empêchant l'exécution des programmes conçus de façon malintentionnée.
La bibliothèque fournie en standard avec Java couvre de nombreux domaines (gestion de collections, accès aux bases de données, interface utilisateur graphique, accès aux fichiers et au réseau, utilisation d'objets distribués, XML…, sans compter toutes les extensions qui s'intègrent sans difficulté à Java) donc la bibliothèque très riche. **[23]**

Java est enfin nativement doté d'un ensemble complet de primitives de gestion du multitâche simplifiant grandement l'écriture de programmes par exemple devant exécuter des requêtes sur une base de données. L'API JDBC (Java DataBase Connectivity), apparue dès la JDK 1.1, permet de développer des applications capables de se connecter à des serveurs de bases de données (SGBD), par le biais d'un pilote.

Nous avons utilisé un tel pilote (com.mysql.jdbc.Driver) de manière à pouvoir se connecter à un serveur mySQL.

2.2. JSP

JSP est une technologie basée sur Java qui permet aux développeurs de générer dynamiquement du code HTML, XML ou tout autre type de page Web, dans laquelle sont introduits des morceaux de code Java nommés "éléments de script".

2.3. MySQL

L'emploi de bases de données était absolument indispensable afin de sauvegarder de manière efficace l'ensemble des tables, qui à long terme peuvent constituer un bloc très volumineux de données.

Nous avons opté pour MySQL (My Structured Query Language) pour plusieurs raisons parmi les quelles la disponibilité de driver pour se connecter à une base de données à partir de Java. Mais aussi, car il offre un système optimisé de gestion de base de données, et ses commandes sont plutôt faciles d'emploi.

2.4. UML

Pour dessiner des diagrammes UML, il existe plusieurs outils disponibles en Open Source (ArgoUML, StarUML, Poséidon, etc.) ou sous forme de plug-in pour Eclipse ou NetBeans. Pour ce qui suit nous avons utilisé ArgoUML. Pour exprimer nos besoins, nous avons utilisé le formalisme UML des cas d'utilisation et des séquences.

3. Modélisation avec UML

3.1. Diagramme des Classes

Le diagramme des classes identifie la structure des classes d'un système, y compris les propriétés et les méthodes de chaque classe. **[24]**

Le diagramme des classes est le diagramme le plus largement répandu dans les spécifications d'UML car il fait abstraction des aspects temporels et **dynamiques**.

Dans ce qui suit, nous avons présenté d'une façon globale les classes qui sont liées avec elles par des relations, selon certaine condition.

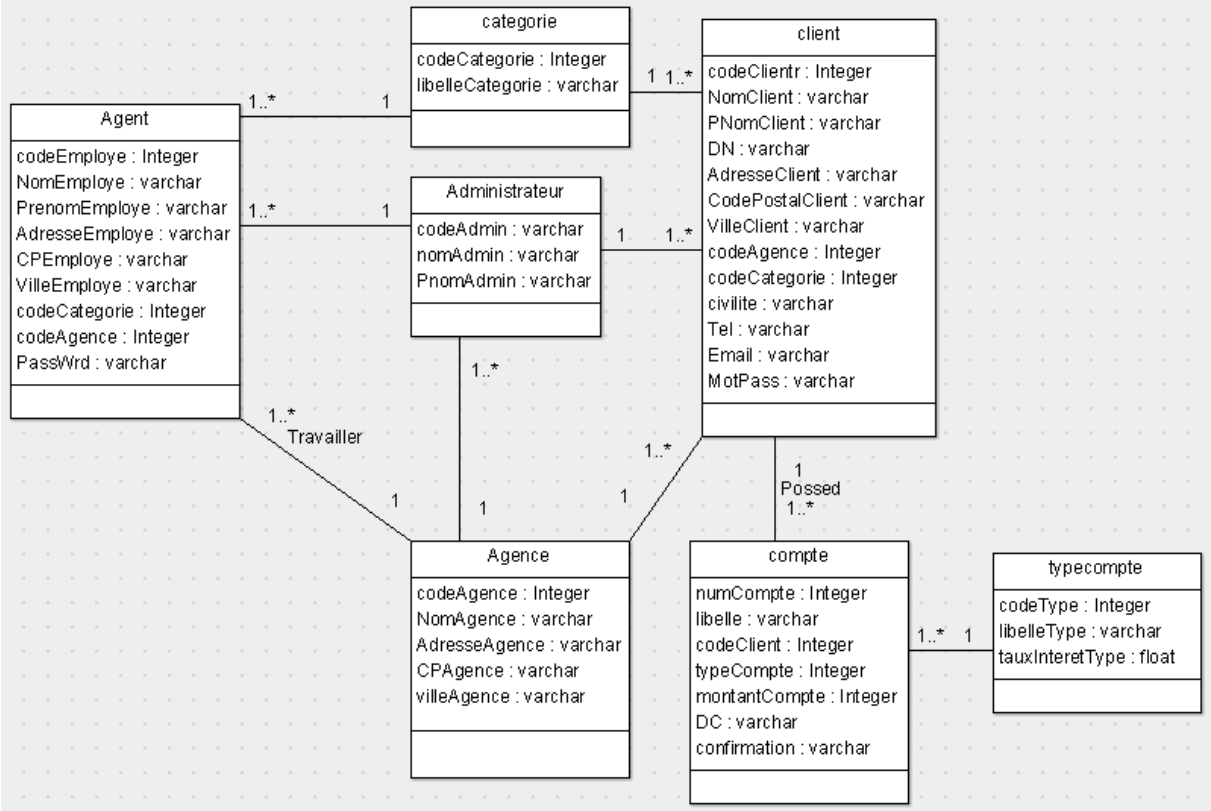

Figure III.1 : Diagramme des classes.

3.2 Diagramme des cas d'utilisation

Les diagrammes des cas d'utilisation identifient 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. **[24]**

Les acteurs humains qui utilisent le système sont les suivants : Agent, Administrateur et Client.

Lorsque le cas d'utilisation est lié à un acteur humain, cela signifie que cet acteur a besoin d'interagir avec le système. Il faut donc lui associer une interface graphique (IHM). Le client utilise le navigateur Web pour accéder au système informatique, alors que les employés et les administrateurs utilisent une application classique déployée sur leurs postes.

Dans le cas où l'acteur serait la base de données il n'y a pas d'interfaces graphiques. Les systèmes communiquent entre eux en échangeant des données.

Les diagrammes des cas d'utilisation se présente comme ci-dessous. Selon chaque acteur.

3.2.1. Premier acteur : Client

Le fonctionnement du premier acteur « Client » est basée sur une architecture de type 3-tiers ou le client utilise une page Web dynamique pour réaliser ses opérations. Un serveur Web qui intègre un conteneur de Servlets vient s'intercaler entre l'IHM et la base de données. Ce modèle présente l'avantage d'intégrer dans cette couche un contrôle d'accès.

La figure suivante, représente le diagramme de cas d'utilisation (Acteur client).

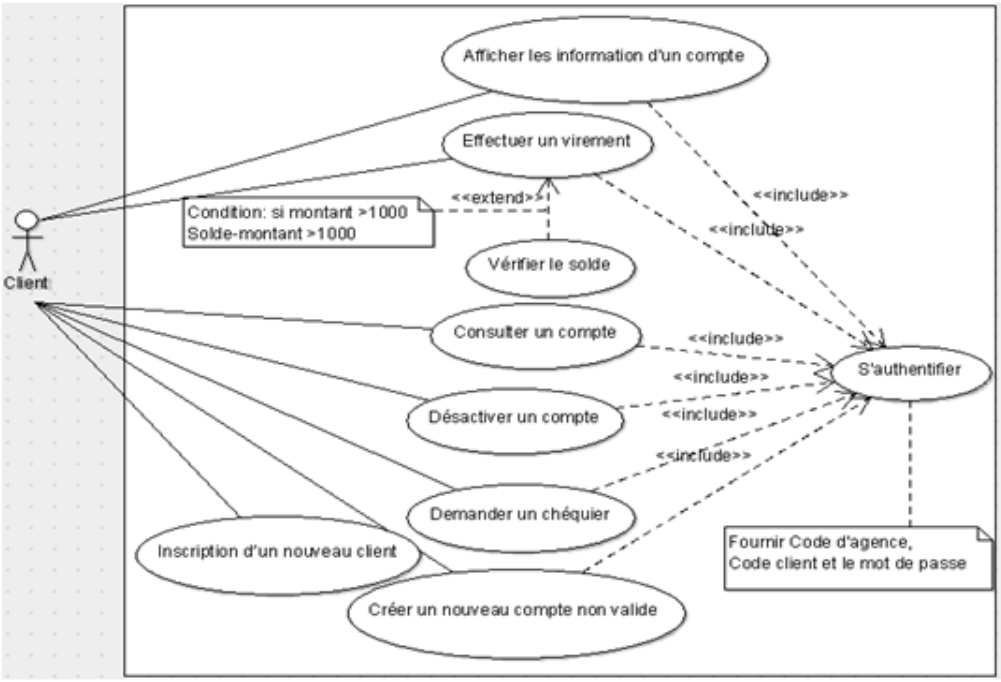

Figure III.2 : Diagramme de cas d'utilisation (Acteur client).

Dans ce qui suit, nous présenterons quelques cas d'utilisations.

- **Cas d'utilisation « Désactiver un compte »**

Le système propose au client de supprimer le compte ou de l'annuler. Dans le cas de la suppression, le système doit attendre une confirmation de l'administrateur avant de le supprimer définitivement.

- **Cas d'utilisation « Créer un nouveau client non valide »**

Permet à un client de s'inscrire dans le système s'il n'existe pas , et pour cela le système lui demande alors de remplir ses coordonnées et ses informations personnels et même les informations concernant son nouveau compte comme libellé, type et solde.

Les éléments caractérisant un client sont les suivants :
• Prénom et nom de famille.
• Date de naissance.
• Numéro de téléphone où l'on peut joindre le client ainsi que son adresse mail.
• Adresse postale.

- **Cas d'utilisation « Consulter le compte »**

Permet à un client de consulter son compte en affichant ses information comme le la date de création, solde, type…etc.

- **Cas d'utilisation « Effectuer un virement »**

Permet d'afficher au client les cordonnées du compte comme le solde, le code du compte et le libellé puis lui donne la main de choisir le compte à créditer qui est soit l'un de ses comptes ou bien un compte d'un autre client et de saisir le montant à virer et le libellé du virement après la validation, le système va vérifier les conditions du virement.

3.2.2 Deuxième acteur : Agent

Le fonctionnement de l'Agent et de l'Administrateur est basé sur une architecture de type 2-tiers vu qu'il n'y a pas d'intermédiaire entre l'IHM et la base de données dans ce cas.

L'idée ici est de faire un contrôle par rôle (Implémenter le **RBAC**) vu la différence en terme de fonctionnalité entre un Agent et un Administrateur. L'agent a pour rôle de gérer le compte des clients qu'il partage avec eux le même code catégorie et le rôle d'administrateur et de gérer la gestion des agences, la gestion des employés, la gestion des clients et la validation des nouveaux clients et des nouveaux comptes.

La figure suivante illustre le diagramme de cas d'utilisation par l'acteur **(Agent).**

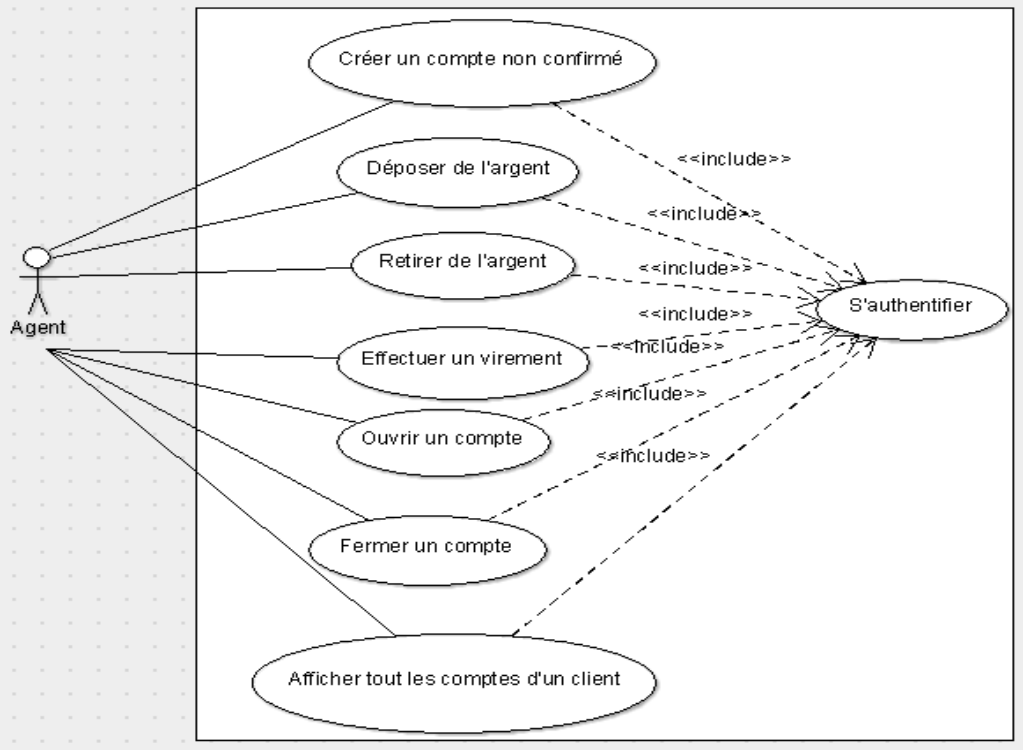

Figure III. 3 : Diagramme de cas d'utilisation (Acteur Agent).

3.2.3. Troisième acteur : Administrateur

La figure suivante illustre le diagramme de cas d'utilisation par l'acteur **(Administrateur).**

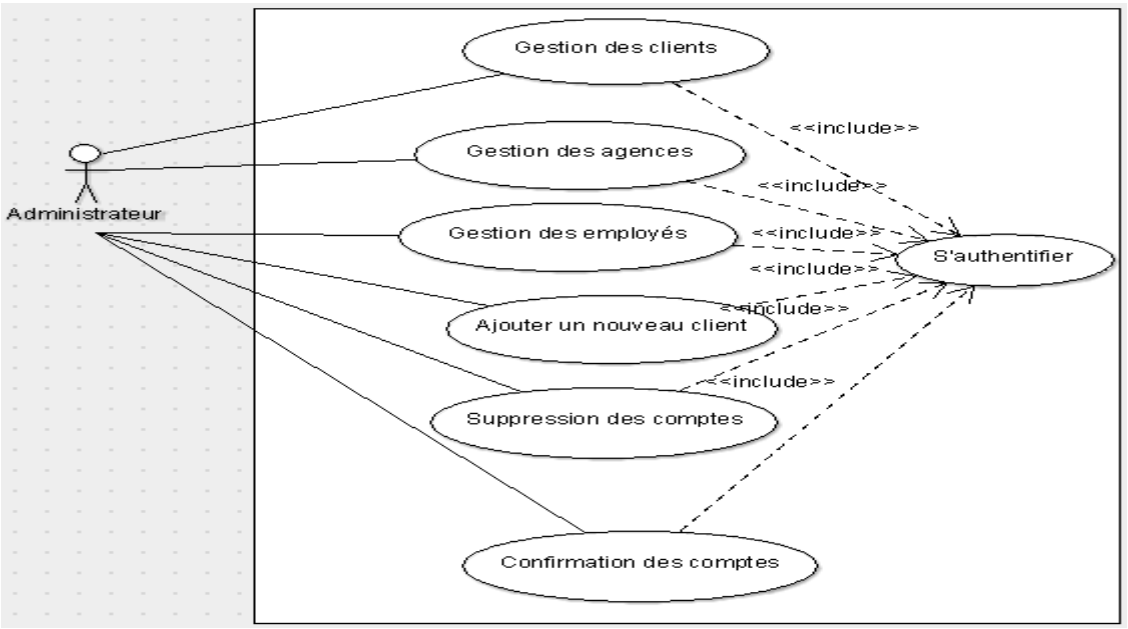

Figure III.4 : Diagramme de cas d'utilisation (Acteur Administrateur).

Dans ce qui suit, nous présenterons quelques cas d'utilisations.

- **Cas d'utilisation « Gestion des clients »**

Permet à un administrateur de créer/modifier/supprimer/rechercher un client et imprimer ses informations.
Notre banque veut pouvoir créer ses clients dans le système à partir des données existantes. Elle souhaite également pouvoir les modifier, les supprimer et les rechercher.

- **Cas d'utilisation « Gestion des agences »**

Permet à un administrateur de modifier/supprimer/rechercher une agence et imprimer les informations, et afficher les clients ou les agents de chaque agence.

- **Cas d'utilisation « Gestion des employés »**

Permet à un administrateur de modifier/supprimer /rechercher des employés et imprimer ses informations.

Chaque cas d'utilisation représenté dans le diagramme précédent doit être complété par un diagramme des séquences.

3.3. Diagramme des séquences

Les diagrammes des séquences documentent les interactions à mettre en œuvre entre les classes pour réaliser un résultat, un cas d'utilisation. UML étant conçu pour la programmation orientée objet, ces communications entre les classes sont reconnues comme des messages. Le diagramme des séquences énumère des objets horizontalement, et le temps verticalement. Il modélise l'exécution des différents messages en fonction du temps. **[24]**

Dans ce qui suit, nous présenterons, les diagrammes des séquences de chaque cas d'utilisation :

3.3.1. Diagramme de séquence de cas d'utilisation « *Authentification* »

Dans ce diagramme, on propose un contrôle d'accès pour authentifier un utilisateur, notre idée est basée sur la désactivation d'un compte après 3 échecs (saisies erronés) d'authentification.

Le digramme est comme suit :

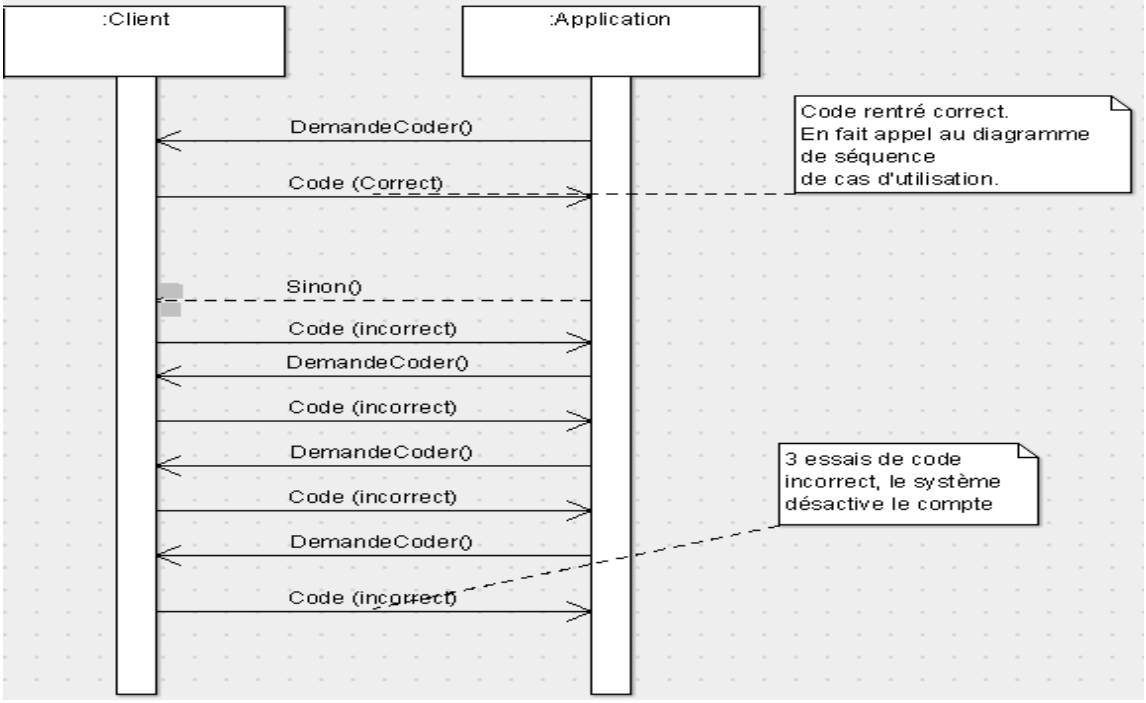

Figure III.5 : Diagramme de séquence de cas d'utilisation « *Authentification* ».

3.3.2. Diagramme de séquence de cas d'utilisation «*Effectuer un virement* »

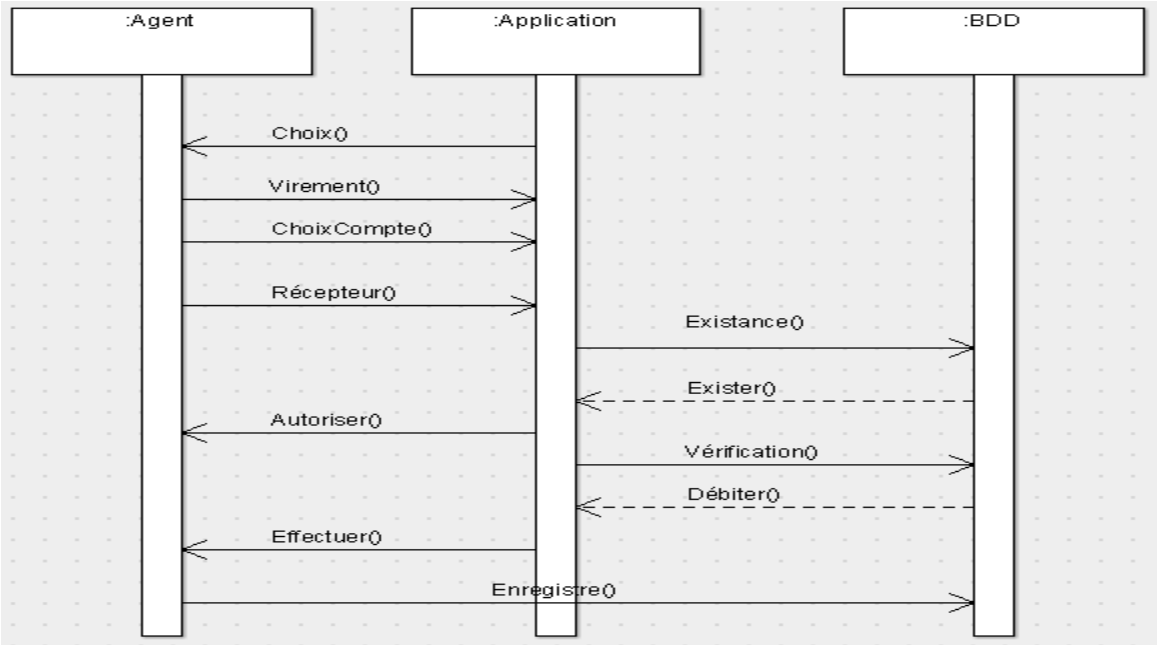

Figure III.6 : Diagramme de séquence de cas d'utilisation «*Effectuer un virement*».

3.3.3. Diagramme de séquence de cas d'utilisation « *Retirer de l'argent* »

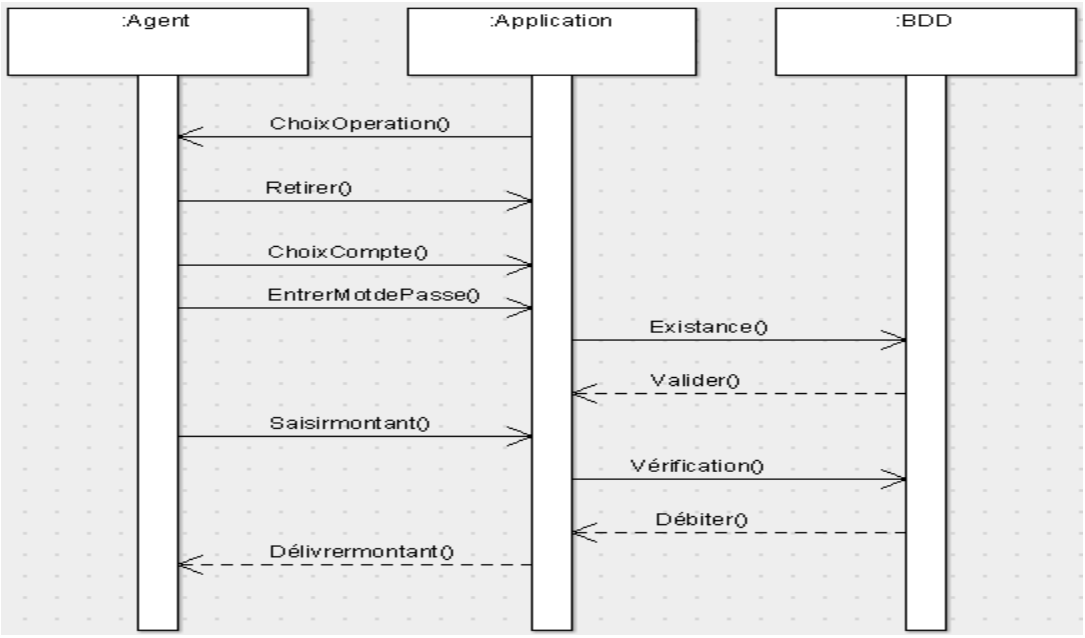

Figure III.7 : Diagramme de séquence de cas d'utilisation «Retirer de l'argent».

4. Conclusion

Dans ce chapitre, nous avons présenté l'étude de cas de l'application gestion bancaire ainsi que les acteurs qui l'utilisent. Le diagramme de cas d'utilisation nous a permis de formaliser les besoins de manière synthétique, puis d'expliciter chaque cas d'utilisation selon le diagramme de séquence. Cette application sera conçue et réalisée dans le chapitre suivant.