Conception	Mocha Bank

Date de mise à jour : 02/06/2021
Version : 1.1

Nom	Delignière Isabelle	Numéro d'auditeur	000378328

Table des matières

1. Objectif du document

Ce document constitue la suite logique des documents d'expression des besoins et d'analyse du projet Mocha Bank et utilise le processus unifié simplifié décrit dans « Enterprise Java with UML » de C. T. Arrington.

Le document de conception logicielle est un document qui fournit une documentation qui sera utilisée pour faciliter le développement de logiciels en donnant les détails sur la façon dont le logiciel doit être construit. Il contient une documentation narrative et graphique de la conception du logiciel pour le projet, y compris des modèles de cas d'utilisation, des diagrammes de séquence, des modèles de collaboration, des modèles de comportement d'objet et d'autres informations sur les exigences de support.

Le but du document de conception logicielle est de présenter une description de la conception d'un système de manière suffisamment systématique pour permettre lors du développement du logiciel de comprendre ce qui doit être construit et comment il devrait être construit. Il expose les informations nécessaires pour donner une description des détails du logiciel et du système à construire.

La 2ème partie du document étudie l'architecture du logiciel et propose un aperçu architectural complet du système, en utilisant un certain nombre de vues architecturales différentes pour décrire différents aspects du système. Puis, la 3ème partie du document aborde les choix technologiques effectués. Ensuite, la 4ème partie du document déroule les cas d'utilisation. Enfin, la 5ème partie du document est consacrée à l'organisation et au regroupement de classes.
Ce document pourra faire l'objet de corrections et/ou être complété par la suite.
Les logiciels StarUML et Visual Paradigm Community Edition ont été utilisés pour réaliser les diagrammes de ce document.

1.1. Points modifiés

Les cas d'utilisations suivants ont été ajoutés en complément :

- Sous-système « gérer le compte bancaire » - Cas de la gestion d'un compte bancaire interne :
  - Exécuter le virement automatique
  - Recréditer le compte après le rejet du virement

Suite à l'application de la règlementation en vigueur, les cas d'utilisations suivants complètent les actions d'authentification dans le cadre de l'authentification multi-facteurs :

- Sous-système « opérations de base » :
  - Authentifier la demande de connexion
- Sous-système « agréger un compte bancaire » :
  - Authentifier la demande d'ajout de compte bancaire externe

1.2. Points non traités

Le choix a été porté sur la pertinence des cas d'utilisation, dans l'idée d'essayer de représenter l'ensemble des objets du système. Ainsi, les éléments suivant n'ont pas été analysés dans ce document et pourront faire l'objet d'un traitement ultérieur.

- Sous-système « opérations de base » :
  - Demander une ouverture de compte
  - Authentifier la demande de connexion
  - Changer de mot de passe
  - Se déconnecter
- Sous-système « gérer les clients » :
  - Créer un client
  - Rechercher un client
  - Mettre à jour les informations d'un client
  - Supprimer un client
  - Créer un compte bancaire
  - Rechercher un compte bancaire
  - Supprimer un compte bancaire
- Sous-système « agréger un compte bancaire » :
  - Ajouter un compte bancaire externe
  - Authentifier la demande d'ajout de compte bancaire externe
  - Supprimer un compte bancaire externe
- Sous-système « gérer le compte bancaire » - Cas de la gestion d'un compte bancaire interne :
  - Consulter l'historique des opérations du compte
  - Faire un dépôt
  - Mettre en place un virement automatique externe
- Sous-système « gérer le compte bancaire » - Cas de la gestion d'un compte bancaire externe agrégé :
  - Consulter le solde d'un compte bancaire agrégé
  - Consulter l'historique des opérations d'un compte bancaire agrégé
  - Faire un dépôt sur un compte bancaire agrégé
  - Effectuer un virement depuis un compte bancaire agrégé
  - Mettre en place un virement automatique externe depuis un compte bancaire agrégé

2. Architecture

2.1. Contraintes techniques

Mocha Bank est un projet personnel, par conséquent, les contraintes techniques sont essentiellement liées au type d'application qu'est Mocha Bank. Toutefois, afin de respecter le cahier des charges défini par les professeurs, des technologies open source telles que les technologies Java Enterprise Edition seront utilisées.

Il s'agit d'une application Web, ainsi, elle utilise le protocole HTTP pour communiquer et doit alors respecter les contraintes techniques minimales imposées par celui-ci.
C'est-à-dire, un serveur Web et un serveur d'application / conteneur de servlets, dans le but de gérer les demandes clients du côté serveur, ainsi qu'un serveur de bases de données pour stocker, extraire, mettre à jour et gérer les données dans une base de données. Celui-ci pourra donner un accès simultané à cette base à plusieurs serveurs Web et utilisateurs.

Dans l'idée d'anticiper toute migration de la base de données, l'application ne doit pas utiliser de langage de requête tel que SQL de manière directe vers la base de données.

Finalement, afin de permettre une accessibilité à un maximum d'utilisateurs, la compatibilité de l'interface utilisateur Web devra être assurée pour les principaux navigateurs Web du marché actuel (Google Chrome, Safari, Mozilla Firefox, Microsoft Edge, Opera).

2.2. Packages et dépendances

La racine des packages sera probablement com.mochabank. Les packages sont définis suivant les couches de l'application définies lors de l'analyse, soit :

- **com.mochabank.service** : les objets de type <<control>> du groupe Service
- **com.mochabank.domain** : les objets de type <<entity>> de la couche métier
- **com.mochabank.dao** : les objets de type <<life cycle>> de la couche persistance

Est également prévu le package suivant :

- **com.mochabank.exception** : les objets de type « Exception ».

2.2.1. Thymeleaf / Vue.js

Bien qu'ils fassent partie de la couche présentation, par défaut, Thymeleaf s'attend à ce que les templates soit placés dans le répertoire src/main/resources /templates. Il est possible de créer des sous-dossiers. Les objets de type <<boundary>> de la couche de présentation seront donc placés ici.

L'utilisation des composants Vue.js côté client nécessite de créer un sous-dossier client et de le remplir avec une application Vue par défaut. La structure de fichiers, simplifiée ressemblera à celle-ci :

```
templates
├── server
└── client
  ├── src
  |   ├── assets
  |   └── components
  └── package.json
```

Pour les fichiers CSS et JavaScript, le répertoire par défaut est src/main/resources/static. Les images seront également placées dans ce répertoire.

2.2.2. DAO/JPA

JPA (Java Persistence API) définit une interface pour conserver les objets Java normaux ou POJO (Plain Old Java Object : bon vieil objet Java) dans une base de données. Cela fournit essentiellement un mappage relationnel-objet.
L'API Java Persistence (JPA) permet donc d'accéder, de conserver et de gérer les données entre les objets / classes Java et une base de données relationnelle. Ainsi, les objets DAO ne seront pas utilisés.

2.2.3. Diagrammes de packages

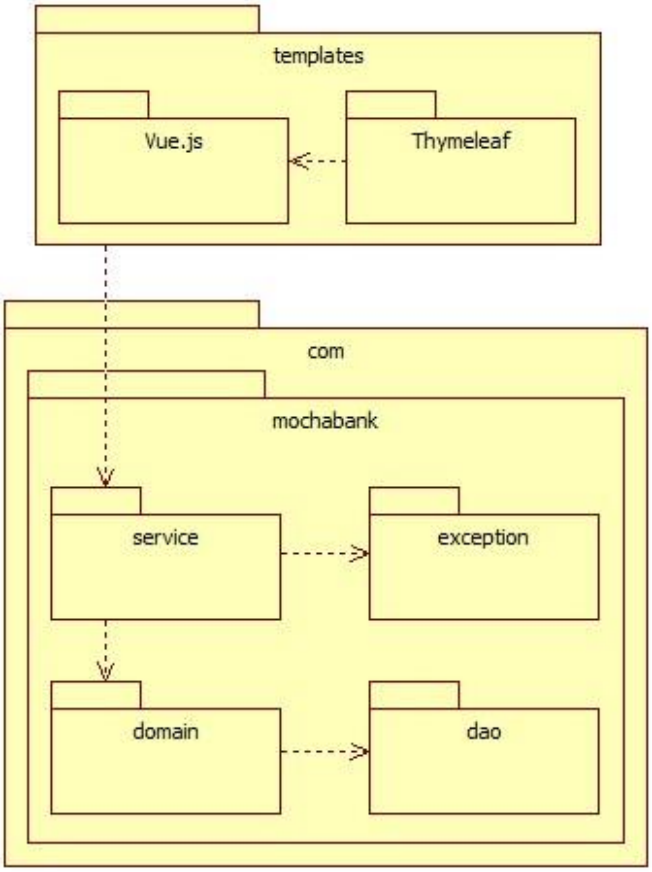

Fig. 1 : diagramme de packages

2.2.4. Packages externes

La gestion des dépendances requises par le projet sera assurée de manière automatisée par Gradle.

Dépendances identifiées :

- **Spring**, notamment le Framework open source Spring, pour construire et définir l'infrastructure d'une application Java, dont il facilite le développement et les tests. Le plugin Spring Boot Gradle fournit un support Spring Boot dans Gradle. Il permet de

regrouper des archives exécutables jar ou war, d'exécuter des applications Spring Boot et d'utiliser la gestion des dépendances fournie par les dépendances spring-boot. Le plug-in Gradle de Spring Boot nécessite Gradle 6 (6.3 ou version ultérieure). Gradle 5.6 est également pris en charge, mais ce support est obsolète et sera supprimé dans une prochaine version.

- **Apache Tomcat**, qui est une implémentation open source des spécifications Jakarta Servlet, Jakarta Server Pages, Jakarta Expression Language, Jakarta WebSocket, Jakarta Annotations et Jakarta Authentication. Ces spécifications font partie de la plateforme Jakarta EE. Il fournit un environnement de serveur Web HTTP dans lequel le code Java peut s'exécuter.
- **JPA**, l'API Java Persistence est une spécification de Java. Elle est utilisée pour conserver les données entre l'objet Java et la base de données relationnelle. JPA agit comme un pont entre les modèles de domaine orientés objet et les systèmes de bases de données relationnelles.
- **Thymeleaf** qui est un moteur de template Java sous licence Apache 2.0, avec des modules pour Spring Framework, idéal pour le développement Web JVM HTML5.
- **Vue.js**, qui est un Framework JavaScript open-source utilisé pour construire des interfaces utilisateur et des applications Web monopages. Le plugin Node Gradle permet d'appeler des tâches NPM (Node Package Manager : gestionnaire de paquets officiel de Node.js) à partir du build Gradle.
- **MySQL**, qui est un système de gestion de bases de données relationnelles sous double licence GPL et propriétaire. mysql-connector-java (MySQL Connector/J : JDBC Type 4 driver for MySQL)
- **JUnit**, qui est un Framework open source pour le développement et l'exécution de tests unitaires automatisables pour le langage de programmation Java.
- **HTTPUnit**, qui émule les parties pertinentes du comportement du navigateur, y compris la soumission de formulaire, JavaScript, l'authentification HTTP de base, les cookies et la redirection automatique des pages, et permet au code de test Java d'examiner les pages renvoyées sous forme de texte, d'un DOM XML ou de conteneurs de formulaires , tableaux et liens. Lorsqu'il est combiné avec un Framework tel que JUnit, il facilite l'écriture des tests qui vérifient très rapidement le fonctionnement d'un site Web.
- **GreenMail**, qui est une suite de tests open source, intuitive et facile à utiliser de serveurs de messagerie à des fins de test

2.3. Déploiement

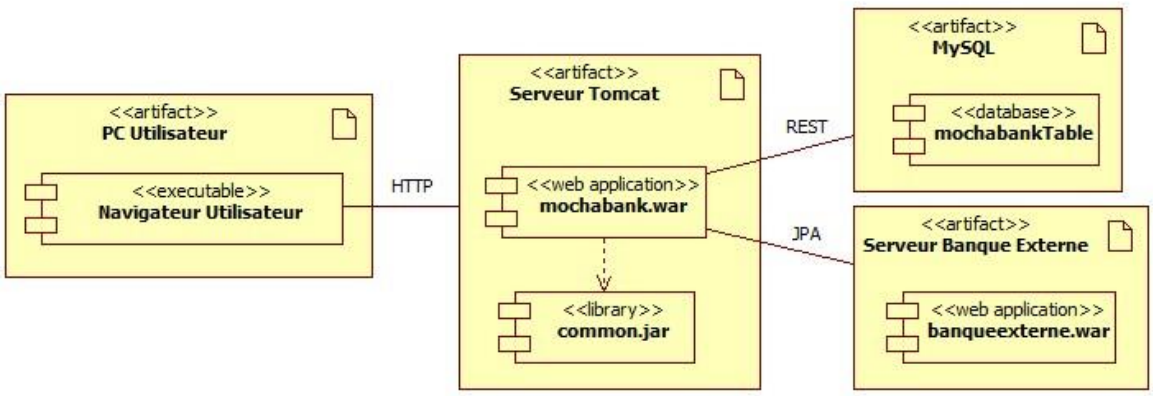

Fig. 2 : diagramme de déploiement

3. Technologies utilisées

3.1. Style architectural

Dans un souci de simplicité, le style architectural « Representational State Transfer » (REST) a été adopté. En effet, REST est la norme la plus logique, la plus efficace et la plus répandue dans la création d'API pour les services Internet. Pour donner une définition simple, REST est n'importe quelle interface entre les systèmes utilisant HTTP pour obtenir des données et générer des opérations sur ces données dans tous les formats possibles, tels que XML et JSON. Ce ne sera par contre probablement pas une application RESTful, c'est-à-dire qui respecte strictement toutes les contraintes architecturales de REST, mais plutôt une application classique se conformant le plus possible aux principes REST.

3.2. Framework d'application

Pour la même raison, le choix a été porté sur le framework Spring, qui est open source, pour construire et définir l'infrastructure de l'application Java, dont il facilitera le développement et les tests. Notamment, Spring Boot qui permet de développer très rapidement de nouvelles applications en utilisant potentiellement toutes les fonctionnalités du framework Spring.

En effet, Spring Boot se charge de configurer l'ensemble de l'application en fonction des dépendances et / ou des annotations Gradle choisies et s'intègre parfaitement dans Gradle. Il permet également de créer une application autonome prête pour la production avec toutes les dépendances (et y compris un serveur d'applications intégré comme Tomcat), conditionnée sous forme de fichier WAR, que l'on peut simplement exécuter via «java -jar <warfile>».

3.3. Sécurité

Pour pouvoir accéder à son espace personnel, un client doit s'authentifier. De même, un employé bancaire doit également s'authentifier afin de gérer les comptes bancaires et les clients de la banque Mocha Bank. L'accès aux différents services de l'application doit ainsi se faire en fonction de droits d'accès définis. Le Framework Spring et ses fonctionnalités de sécurité seront utilisés. Spring permet notamment de gérer les authentifications et autorisations.

Le service d'agrégation de compte bancaire externe nécessite une attention particulière. En effet, il est fondamental de s'assurer que l'utilisateur dispose bien des droits sur le compte bancaire externe. La DSP2 (Directive européenne relative aux services de paiement), en vigueur depuis début 2018, donne un cadre légal aux agrégateurs de comptes. Un portail d'API tests des banques dédié aux développeurs doit être accessible pour éviter d'avoir recours au « screen scraping ». De plus, la règlementation impose la mise en place d'une solution d'authentification forte pour valider les paiements et sceller les transactions sensibles. L'authentification forte est, en sécurité des systèmes d'information, une procédure d'identification qui requiert la concaténation d'au moins deux facteurs d'authentification.

Actuellement, concernant l'ajout d'un compte bancaire externe, les agrégateurs utilisent l'identifiant de la banque, le numéro d'IBAN (International Bank Account Number), les codes

d'accès à l'espace client, la date de naissance etc… Certain agrégateurs s'affranchissent du risque d'opérations de virement malveillantes en n'autorisant que les opérations de consultation des soldes et d'historiques des opérations. D'autres, utilisent en plus l'authentification multi-facteurs (MFA) avec l'envoi d'un code par SMS sur le téléphone mobile de l'utilisateur comme pour les systèmes de paiement en ligne.

L'architecture MFA nécessite une passerelle modem SMS en plus de HTTP pour pouvoir envoyer un SMS ainsi qu'un opérateur télécom. Il est possible d'utiliser une passerelle open source comme Jasmin, PlaySMS, Kannel, ou encore Kalkun, par exemple ou de la créer soi-même mais il reste nécessaire de souscrire à un contrat auprès d'un opérateur télécom. Une solution comme Okta OAuth 2.0, qui permet d'authentifier 15,000 utilisateurs par mois « gratuitement », pourrait également être intégrée. Cette dernière solution ne s'appliquerait pas dans une banque qui ferait probablement plutôt appel à un opérateur dans ce cas.

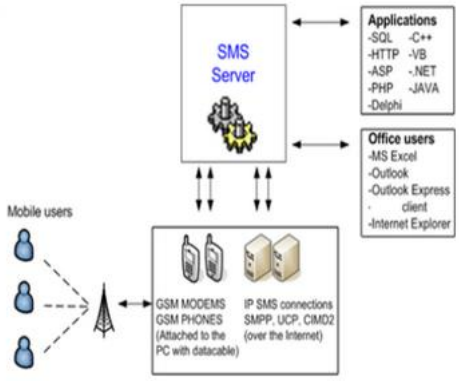

Fig. 3 : SMS gateway system architecture

Plutôt que d'intégrer une solution gratuite où l'on est peut-être le produit, ou d'utiliser un « mock » pour simuler l'envoi de SMS, par exemple vers un site tiers, un fichier de log ou un logiciel externe, il a été choisi d'utiliser simplement l'envoi d'un email. Spring Framework fournit une bibliothèque utilitaire utile pour l'envoi d'e-mails qui protège des spécificités du système de messagerie sous-jacent et est responsable de la gestion des ressources de bas niveau pour le compte du client et sera donc utilisé dans ce cadre.

Enfin, Spring permettra également d'assurer le chiffrement des données. Le module Spring Security Crypto prend en charge le chiffrement symétrique, la génération de clés et le codage de mot de passe.

3.4. Serveur Web

L'utilisation du logiciel Apache Tomcat, qui est une implémentation open source des technologies Java Servlet, JavaServer Pages, Java Expression Language et Java WebSocket, a été adoptée. Il offre ainsi un environnement de serveur Web « pur Java » pour exécuter des applications basées sur le langage de programmation Java. Apache Tomcat est une application légère. En effet, il n'offre que les fonctionnalités les plus basiques nécessaires pour exécuter un serveur, ce qui signifie qu'il fournit des temps de chargement et de redéploiement relativement rapides. Il permet également de profiter d'un cycle de développement rapide.

3.5. Stockage des données

Le système de gestion de base de données relationnelle basé sur SQL (Structured Query Language) MySQL a été choisi. En effet, Il est open source et fournit un support complet pour tous les besoins de développement d'applications. MySQL fournit également des connecteurs et des pilotes (ODBC, JDBC, etc.) qui permettent à toutes les formes d'applications d'utiliser MySQL comme serveur de gestion de données. Les connexions seront faites grâce au driver JDBC. Enfin, il s'agit d'une base de données persistante qui conserve les données dans la mémoire physique. Les données seront donc disponibles même si le serveur de base de données ne l'est pas.

3.6. Couche de persistance

Dans l'idée de conserver une portabilité de la base de données et de respecter ainsi les contraintes techniques, l'API « Java Persistence » (JPA), qui est une spécification d'interface de programmation d'application Java et qui décrit la gestion des données relationnelles dans les applications utilisant Java Platform, Standard Edition et Java Platform, Enterprise Edition / Jakarta EE, sera utilisée.
En effet, JPA permet de charger et d'enregistrer des objets et des graphiques Java sans aucun Langage de manipulation de données. JPQL permet d'exprimer les requêtes en termes d'entités Java plutôt qu'en termes de tables et colonnes SQL (natives).
Le Framework Spring ayant été choisi, Spring Data JPA sera utilisée pour gérer les interactions de la base de données. L'objectif de l'abstraction de Spring Data repository est de réduire considérablement la quantité de code standard requis pour implémenter des couches d'accès aux données pour divers stockages de persistance. Spring fournit ainsi l'interface CrudRepositor. Pour l'utiliser, il faut créer une interface en étendant CrudRepository pour un type spécifique. CrudRepository dispose des fonctionnalités CRUD (Create, Read, Update, Delete) sophistiquées pour la classe d'entité gérée, et contient donc des méthodes telles que save, findById, delete, count etc. Si des méthodes supplémentaires sont nécessaires, il faut les déclarer dans l'interface héritée.

3.7. Couche métier

Afin d'augmenter la lisibilité et la réutilisation, les POJO seront utilisés pour la couche métier.

3.8. Couche service

Les POJO seront utilisés pour implémenter la couche service pour les mêmes raisons que pour la couche métier. L'idée est de permettre l'évolution des services de manière simple.

3.9. Couche présentation

Le moteur de templates Thymeleaf, sous licence Apache 2.0, écrit en Java et pouvant générer du XML/XHTML/HTML5, a été sélectionné pour générer des pages web dynamiques.

L'objectif principal de Thymeleaf est d'être utilisé dans un environnement web pour la génération de vue pour les applications Web basées sur le modèle MVC (Modèle-vue-contrôleur). Thymeleaf facilite la lecture du code ainsi que l'intégration à l'aide des modules Spring Framework.
Le Framework Vue.js sera utilisé en complément pour les composants JavaScript côté client. Vue.js peut être utilisé pour créer des Single Page Applications, mais il prend aussi spécifiquement en charge l'exportation de composants à consommer en dehors d'une Single Page Application.
L'idée est de créer une application Spring Boot avec le moteur de templates côté serveur Thymeleaf et une bibliothèque de composants JavaScript qui fournit un composant JavaScript construit avec NPM et Vue. Le résultat sera une application hybride qui permet au moteur de templates côté serveur de créer des pages HTML statiques tout en incluant des composants JavaScript qui offrent plus d'interactivité.
L'utilisation des composants Vue.js côté client dans les pages HTML générées avec le moteur de création de modèles côté serveur Thymeleaf permet de prévisualiser les composants Javascript sans démarrer l'application côté serveur, d'écrire et exécuter des tests pour ces composants Javascript, d'inclure des composants Javascript sélectionnés dans une page HTML rendue par le serveur sans les charger tous, de réduire le Javascript, et d'intégrer la construction des composants Javascript avec la construction de l'application côté serveur.

3.10. Environnement de développement

Les logiciels suivants ont été choisis pour développer l'application :

- **Eclipse** : IDE (Integrated Development Environment) sous licence libre.
- **Spring Tools 4 pour Eclipse** : outil open source pour le développement d'applications d'entreprise basées sur Spring.
- **WampServer 3.2.0** : une plate-forme de développement Web sous Windows qui permet de créer des applications Web dynamiques avec Apache2, PHP, MySQL et MariaDB, disponible gratuitement (sous licence GPML) dans les versions 32 et 64 bits.
- **Node.js** : est un runtime JavaScript basé sur le moteur JavaScript V8 de Chrome. Il est nécessaire d'installer Node.js pour prendre en charge l'environnement de développement Vue.js. Une fois que Node.js est installé, il est possible d'installer la Vue CLI, cela permet d'obtenir la commande vue à utiliser pour créer le projet Vue.
- **Tomcat 9.0.26** : serveur web pour des applications java.
- **Gradle** : système d'automatisation de production open source qui s'appuie sur les concepts d'Apache Ant et d'Apache Maven et introduit un langage spécifique au domaine basé sur Groovy au lieu du formulaire XML utilisé par Maven pour déclarer la configuration du projet. Gradle gère les dépendances du projet, la compilation et la production de composants distribuables (artifacts).

3.11. Test unitaire

Les tests unitaires seront spécifiés et développés au cours de la conception de l'application. JUnit, HTTPUnit et GreenMail seront utilisés conjointement.

JUnit est Framework de test unitaire pour le langage de programmation Java.

HttpUnit est un framework de test de logiciels open source utilisé pour effectuer des tests de sites Web sans avoir besoin d'un navigateur Web. HttpUnit prend en charge la soumission de formulaires HTML, JavaScript, l'authentification d'accès de base HTTP, la redirection de page automatique et les cookies. Écrit en Java, HttpUnit permet au code de test Java de traiter les pages renvoyées sous forme de texte, de DOM XML ou de conteneurs de formulaires, de tableaux et de liens.

HttpUnit est bien adapté pour être utilisé en combinaison avec JUnit, afin d'écrire facilement des tests qui vérifient le bon comportement d'un site Web.

GreenMail est une suite de tests, open source, de serveurs de messagerie à des fins de test. Elle permet de réaliser des tests unitaires sur le code qui doit envoyer des e-mails de façon succincte et efficace (sans interrogation) d'attendre l'arrivée des messages, de les récupérer, de les vérifier et de les modifier.

4. Cas d'utilisation

4.1. Sous-système « opérations de base »

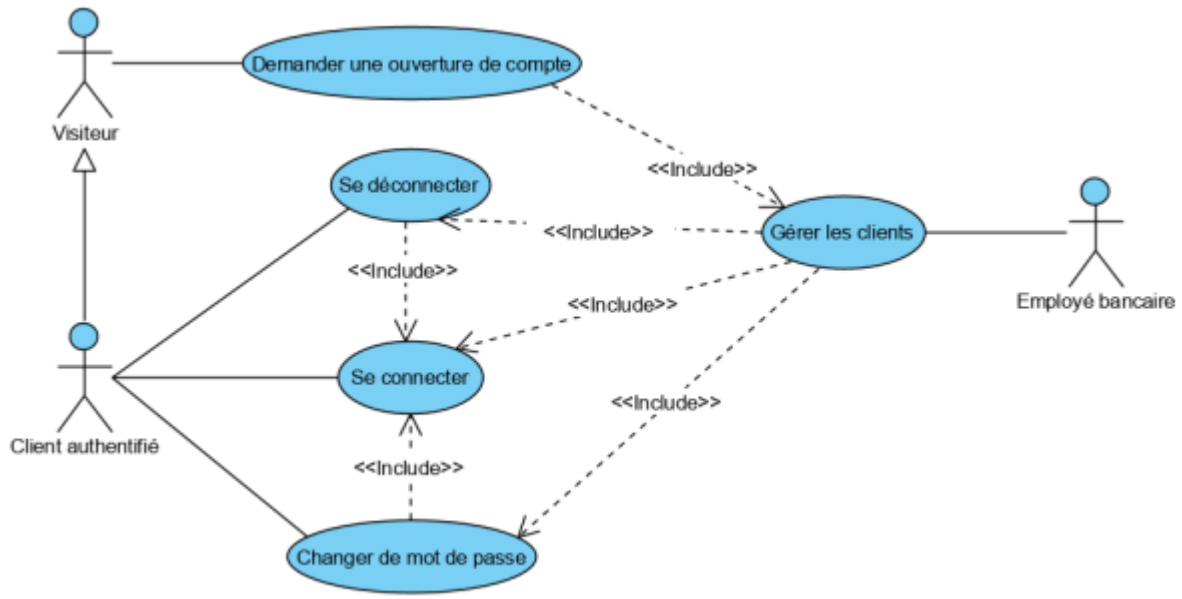

Fig. 4 : diagramme de cas d'utilisation sous-système « opérations de base »

4.1.1. Se connecter

4.1.1.1. Liste des objets candidats

- <<Entity>>

User : Objet représentant un utilisateur.
Customer : Objet représentant un client.
BankClerk : Objet représentant un employé bancaire.
Address : Objet représentant une adresse postale.

- <<Boundary >>

loginForm.html : page de formulaire pour le login par laquelle l'utilisateur (le client ou l'employé bancaire) va interagir pour s'authentifier.
accueil.html : page d'accueil de l'utilisateur authentifié.
AuthenticationController : servlet servant à la gestion de l'authentification.

- <<Control>>

LoginService : Objet chargé de la logique métier d'authentification. Il gère l'affichage de la page logingForm.html ainsi que la validation de l'authentification.
UserService : Objet chargé de la logique métier utilisateur.

- << Life cycle>>

UserRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité User persistantes, pour les rechercher par leur clé primaire et pour interroger.
CustomerRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Customer persistantes, pour les rechercher par leur clé primaire et pour interroger.
BankClerkRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité BankClerk persistantes, pour les rechercher par leur clé primaire et pour interroger.

4.1.1.2. Description des interactions entre objets

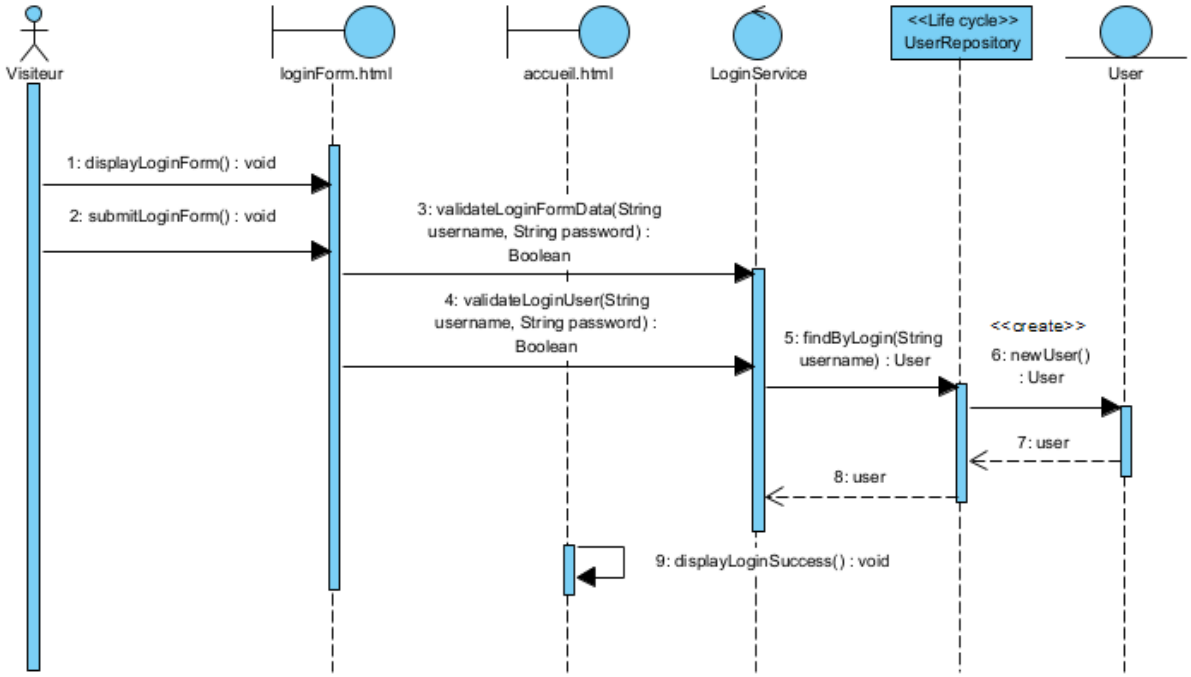

Fig. 5 : diagramme de séquences « se connecter »

4.1.1.3. Description des classes

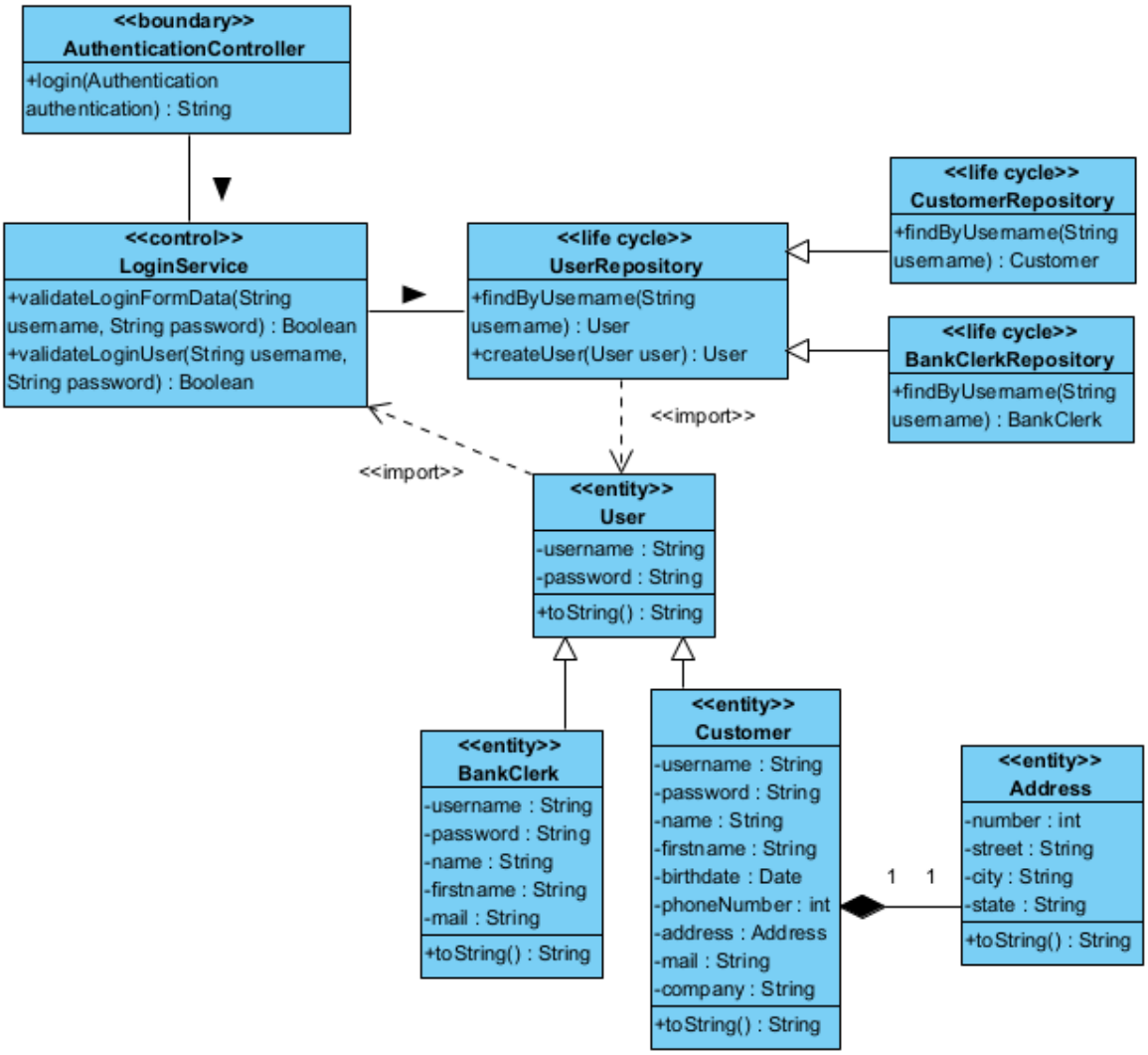

Fig. 6 : diagramme de classes « se connecter »

4.2. Sous-système « gérer le compte bancaire »

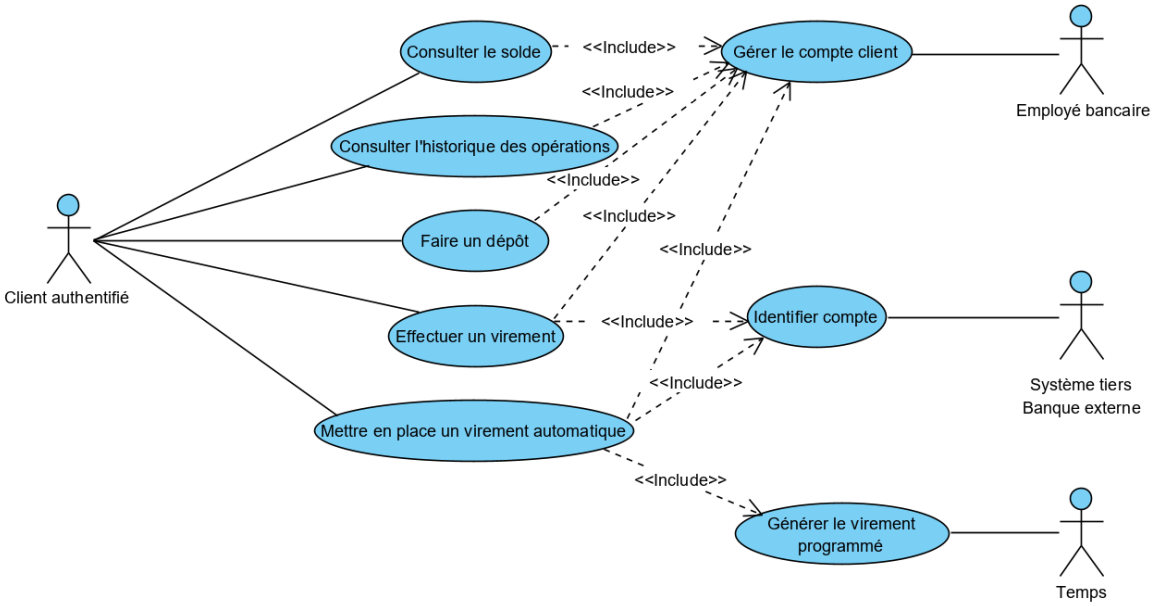

Fig. 7 : diagramme de cas d'utilisation sous-système « gérer le compte bancaire »

4.2.1. Consulter le solde du compte bancaire

4.2.1.1. Liste des objets candidats

- <<Entity>>

Customer : Objet représentant un client.
Address : Objet représentant une adresse postale.
Account : Objet représentant un compte bancaire.
Transaction : Objet représentant une opération bancaire.
Operation : Objet représentant une opération bancaire de base.

- << Boundary >>

checkbalance.html : page affichant la balance du compte bancaire.
AccountManagementController : servlet servant à la gestion des comptes bancaires.

- <<Control>>

AccountManagementService : Objet chargé de la logique métier de gestion de compte bancaire. Il gère l'affichage de l'objet accountManagement.html ainsi que les opérations de gestion de compte (consulter le solde, consulter l'historique des opérations, faire un dépôt, effectuer un virement interne/externe, mettre en place un virement automatique interne/externe).

- << Life cycle >>

CustomerRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Customer persistantes, pour les rechercher par leur clé primaire et pour interroger.
AccountRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Account persistantes, pour les rechercher par leur clé primaire et pour interroger.
TransactionRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Transaction persistantes, pour les rechercher par leur clé primaire et pour interroger.
OperationRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Operation persistantes, pour les rechercher par leur clé primaire et pour interroger.

4.2.1.2. Description des interactions entre objets

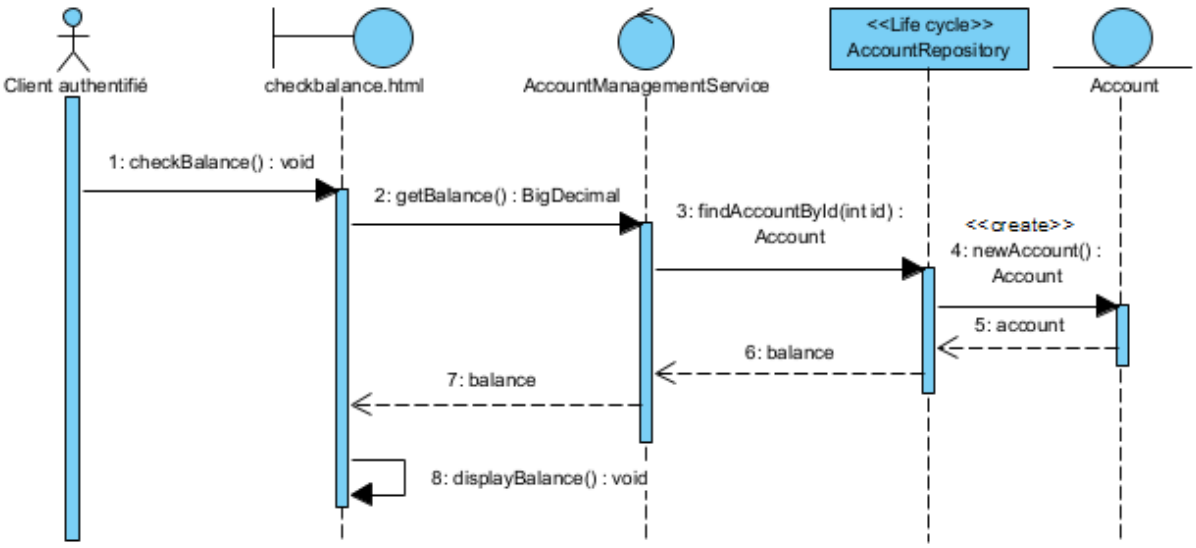

Fig. 8 : diagramme de séquences « consulter le solde du compte bancaire »

4.2.1.3. Description des classes

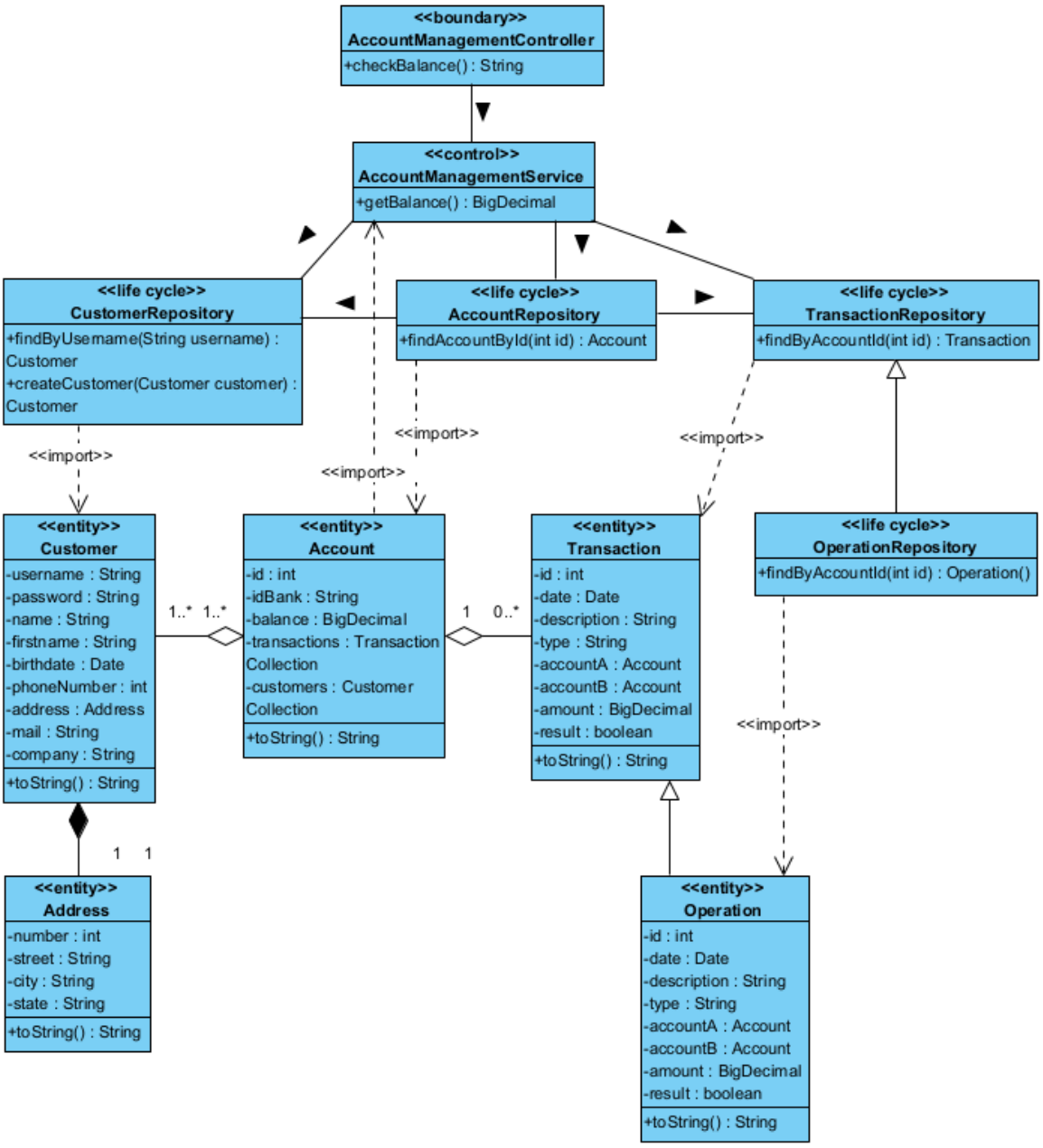

Fig. 9 : diagramme de classes « consulter le solde du compte bancaire »

4.2.2. Effectuer un virement interne

Cette opération est immédiate. Si le client choisit une date ultérieure pour effectuer un virement, alors il s'agit du cas d'utilisation « mettre en place un virement automatique interne ».

4.2.2.1. Liste des objets candidats

- <<Entity>>

Customer : Objet représentant un client.
Address : Objet représentant une adresse postale.
Account : Objet représentant un compte bancaire.
Transaction : Objet représentant une opération bancaire.
Operation : Objet représentant une opération bancaire de base.

- <<Boundary>>

transferform.html : page de formulaire pour le virement par laquelle l'utilisateur (le client ou l'employé bancaire) va interagir pour effectuer un virement.
result.html : page affichant le résultat de l'opération bancaire effectuée, ici, un virement interne.
AccountManagementController : servlet servant à la gestion des comptes bancaires.

- <<Control>>

AccountManagementService : Objet chargé de la logique métier de gestion de compte bancaire. Il gère l'affichage de l'objet accountManagement.html ainsi que les opérations de gestion de compte (consulter le solde, consulter l'historique des opérations, faire un dépôt, effectuer un virement interne/externe, mettre en place un virement automatique interne/externe).

- << Life cycle>>

CustomerRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Customer persistantes, pour les rechercher par leur clé primaire et pour interroger.
AccountRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Account persistantes, pour les rechercher par leur clé primaire et pour interroger.
TransactionRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Transaction persistantes, pour les rechercher par leur clé primaire et pour interroger.
OperationRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Operation persistantes, pour les rechercher par leur clé primaire et pour interroger.

4.2.2.2. Description des interactions entre objets

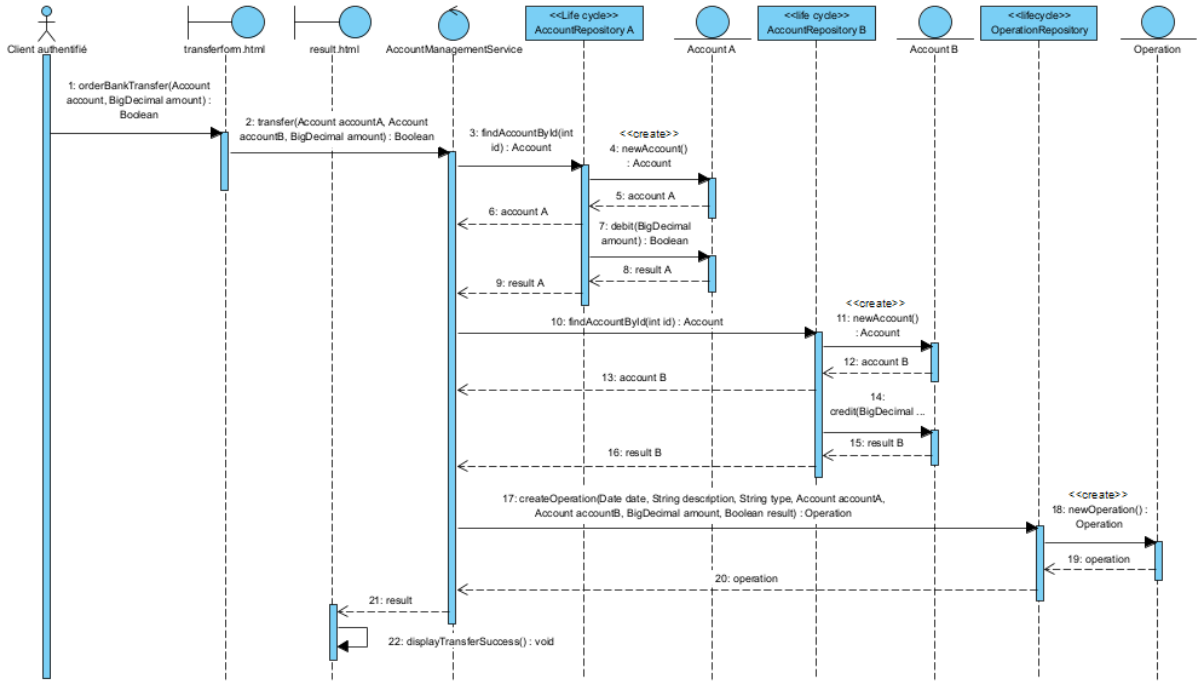

Fig. 10 : diagramme de séquences « effectuer un virement interne »

4.2.2.3. Description des classes

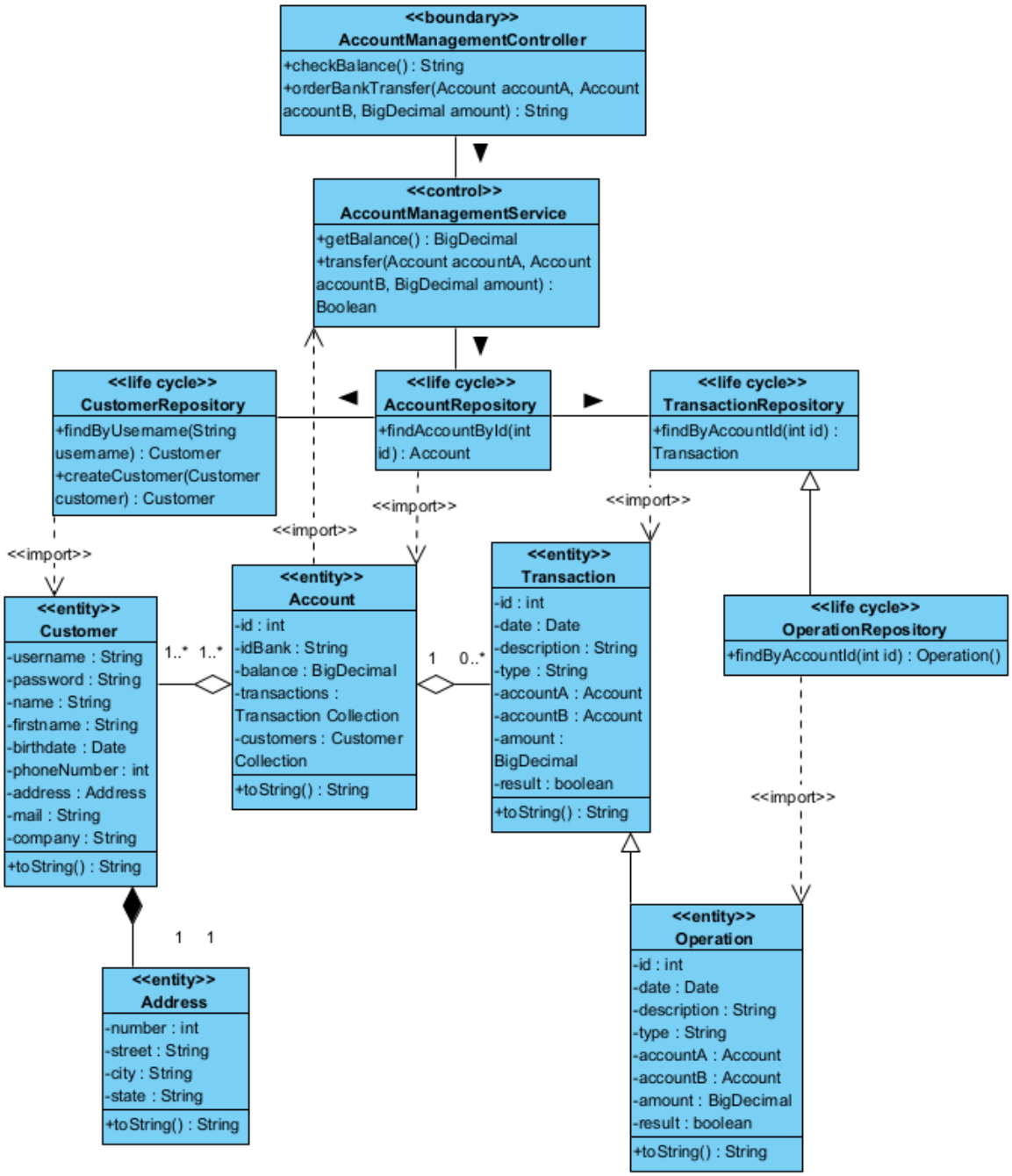

Fig. 11 : diagramme de classes « effectuer un virement interne »

4.2.3. Effectuer un virement externe

Cette opération est immédiate. Si le client choisit une date ultérieure pour effectuer un virement, alors il s'agit du cas d'utilisation « mettre en place un virement automatique externe ».

4.2.3.1. Liste des objets candidats

- <<Entity>>

Customer : Objet représentant un client.
Address : Objet représentant une adresse postale.
Account : Objet représentant un compte bancaire.
Transaction : Objet représentant une opération bancaire.
Operation : Objet représentant une opération bancaire de base.

- <<Boundary>>

transferform.html : page de formulaire pour le virement par laquelle l'utilisateur (le client ou l'employé bancaire) va interagir pour effectuer un virement.
result.html : page affichant le résultat de l'opération bancaire effectuée, ici, un virement externe.
AccountManagementController : servlet servant à la gestion des comptes bancaires.

- <<Control>>

AccountManagementService : Objet chargé de la logique métier de gestion de compte bancaire. Il gère l'affichage de l'objet accountManagement.html ainsi que les opérations de gestion de compte (consulter le solde, consulter l'historique des opérations, faire un dépôt, effectuer un virement interne/externe, mettre en place un virement automatique interne/externe).

- << Life cycle>>

CustomerRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Customer persistantes, pour les rechercher par leur clé primaire et pour interroger.
AccountRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Account persistantes, pour les rechercher par leur clé primaire et pour interroger.
TransactionRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Transaction persistantes, pour les rechercher par leur clé primaire et pour interroger.
OperationRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Operation persistantes, pour les rechercher par leur clé primaire et pour interroger.

4.2.3.2. Description des interactions entre objets

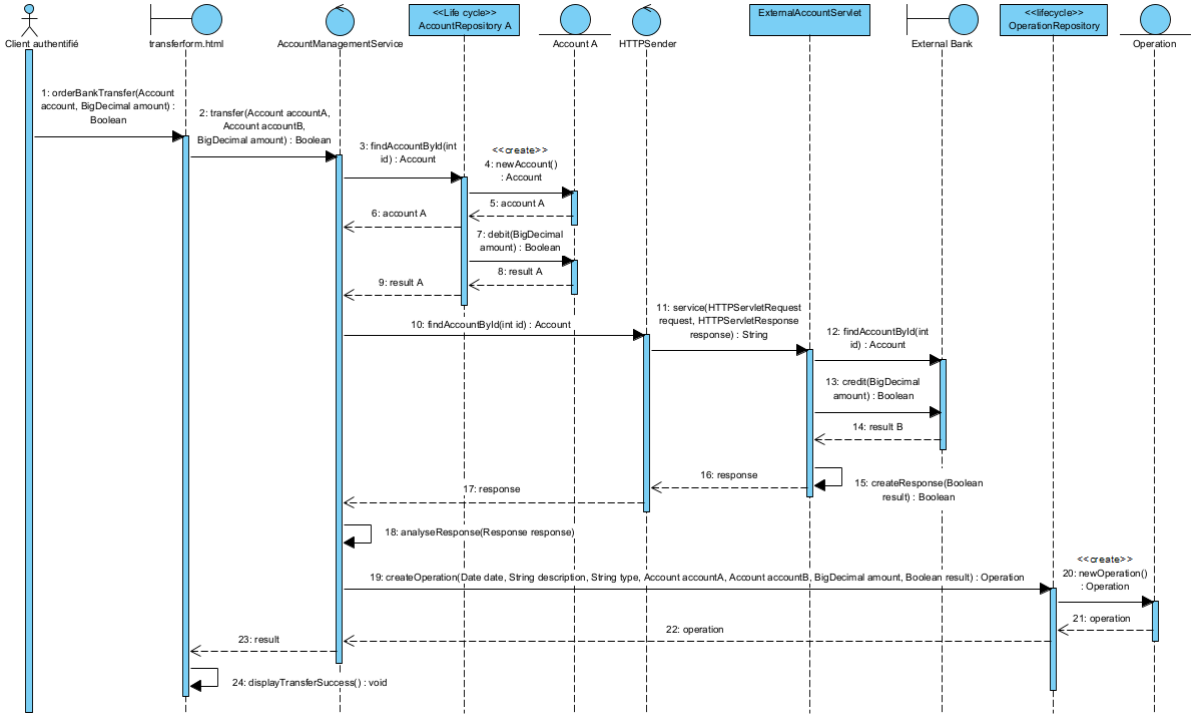

Fig. 12 : diagramme de séquences « effectuer un virement externe »

4.2.3.3. Description des classes

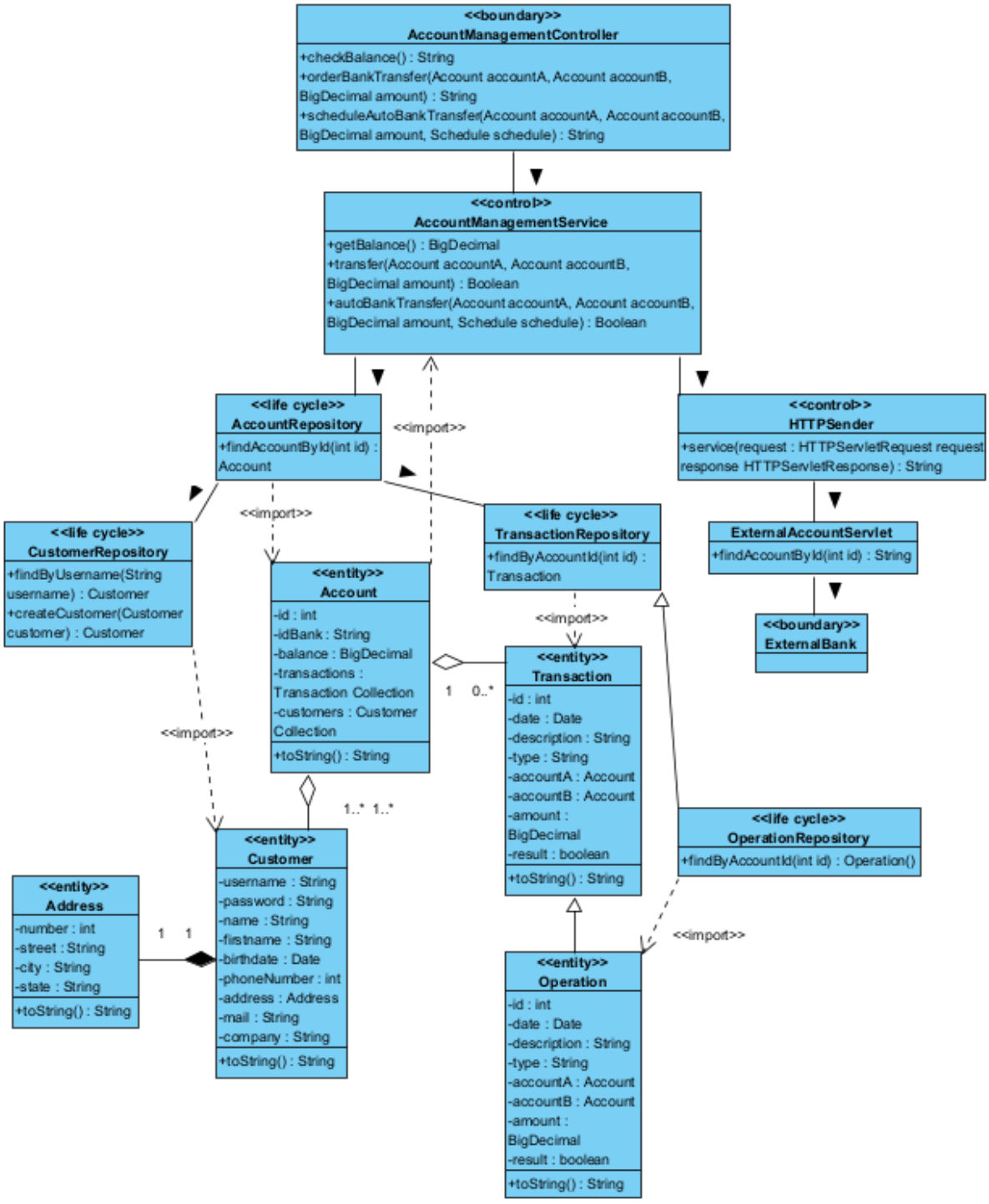

Fig. 13 : diagramme de classes « effectuer un virement externe »

4.2.4. Mettre en place un virement automatique interne

4.2.4.1. Liste des objets candidats

- <<Entity>>

Customer : Objet représentant un client.
Address : Objet représentant une adresse postale.
Account : Objet représentant un compte bancaire.
Transaction : Objet représentant une opération bancaire.
ScheduledOperation : Objet représentant une opération bancaire programmée.

- <<Boundary>>

transferform.html : page de formulaire pour le virement par laquelle l'utilisateur (le client ou l'employé bancaire) va interagir pour effectuer un virement.
result.html : page affichant le résultat de l'opération bancaire effectuée, ici, la mise en place d'un virement automatique.
AccountManagementController : servlet servant à la gestion des comptes bancaires.

- <<Control>>

AccountManagementService : Objet chargé de la logique métier de gestion de compte bancaire. Il gère l'affichage de l'objet accountManagement.html ainsi que les opérations de gestion de compte (consulter le solde, consulter l'historique des opérations, faire un dépôt, effectuer un virement interne/externe, mettre en place un virement automatique interne/externe).

- << Life cycle>>

CustomerRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Customer persistantes, pour les rechercher par leur clé primaire et pour interroger.
AccountRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Account persistantes, pour les rechercher par leur clé primaire et pour interroger.
TransactionRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Transaction persistantes, pour les rechercher par leur clé primaire et pour interroger.
ScheduledOperationRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité ScheduledOperation persistantes, pour les rechercher par leur clé primaire et pour interroger.

4.2.4.2. Description des interactions entre objets

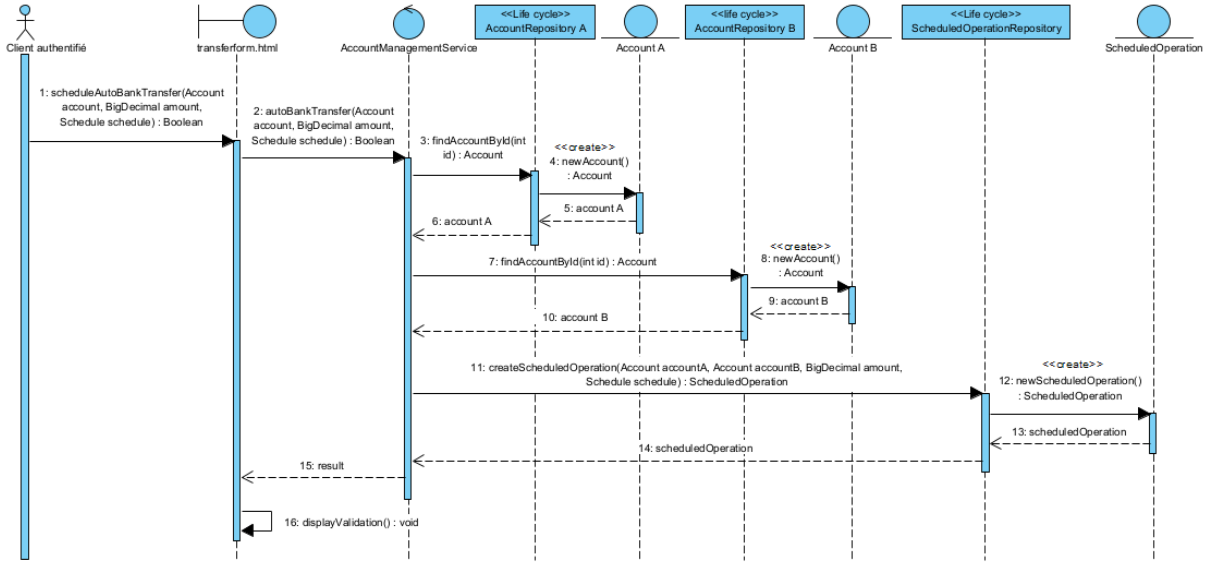

Fig. 14 : diagramme de séquences « mettre en place un virement automatique interne »

4.2.4.3. Description des classes

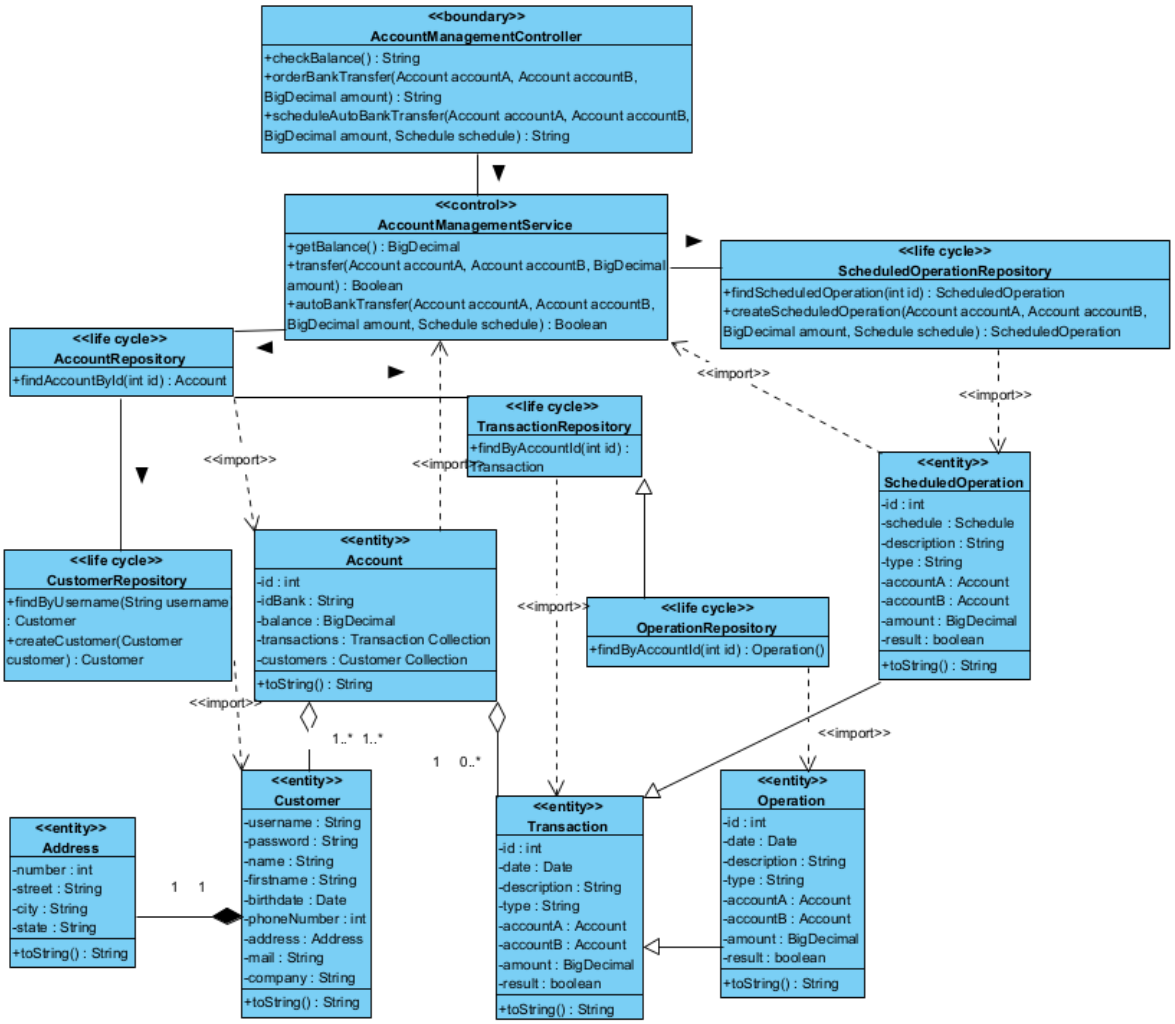

Fig. 15 : diagramme de classes « mettre en place un virement automatique interne »

4.2.5. Exécuter le virement automatique

4.2.5.1. Liste des objets candidats

- <<Entity>>

Customer : Objet représentant un client.
Address : Objet représentant une adresse postale.
Account : Objet représentant un compte bancaire.
Transaction : Objet représentant une opération bancaire.
Operation : Objet représentant une opération bancaire de base.
ScheduledOperation : Objet représentant une opération bancaire programmée.

- <<Boundary>>

checkoperations.html : page affichant l'historique des opérations du compte bancaire. Suite à l'exécution du virement automatique par le système, celui-ci sera visible dans l'historique des opérations.
AccountManagementController : servlet servant à la gestion des comptes bancaires.

- <<Control>>

AccountManagementService : Objet chargé de la logique métier de gestion de compte bancaire. Il gère l'affichage de l'objet accountManagement.html ainsi que les opérations de gestion de compte (consulter le solde, consulter l'historique des opérations, faire un dépôt, effectuer un virement interne/externe, mettre en place un virement automatique interne/externe).

- << Life cycle>>

CustomerRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Customer persistantes, pour les rechercher par leur clé primaire et pour interroger.
AccountRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Account persistantes, pour les rechercher par leur clé primaire et pour interroger.
TransactionRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Transaction persistantes, pour les rechercher par leur clé primaire et pour interroger.
OperationRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Operation persistantes, pour les rechercher par leur clé primaire et pour interroger.
ScheduledOperationRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité ScheduledOperation persistantes, pour les rechercher par leur clé primaire et pour interroger.

N.B. : En cas de rejet du virement automatique, si le compte a été débité, le cas d'utilisation « recréditer le compte après le rejet du virement » sera exécuté et l'opération sera visible dans l'historique des opérations.

4.2.5.1. Description des interactions entre objets

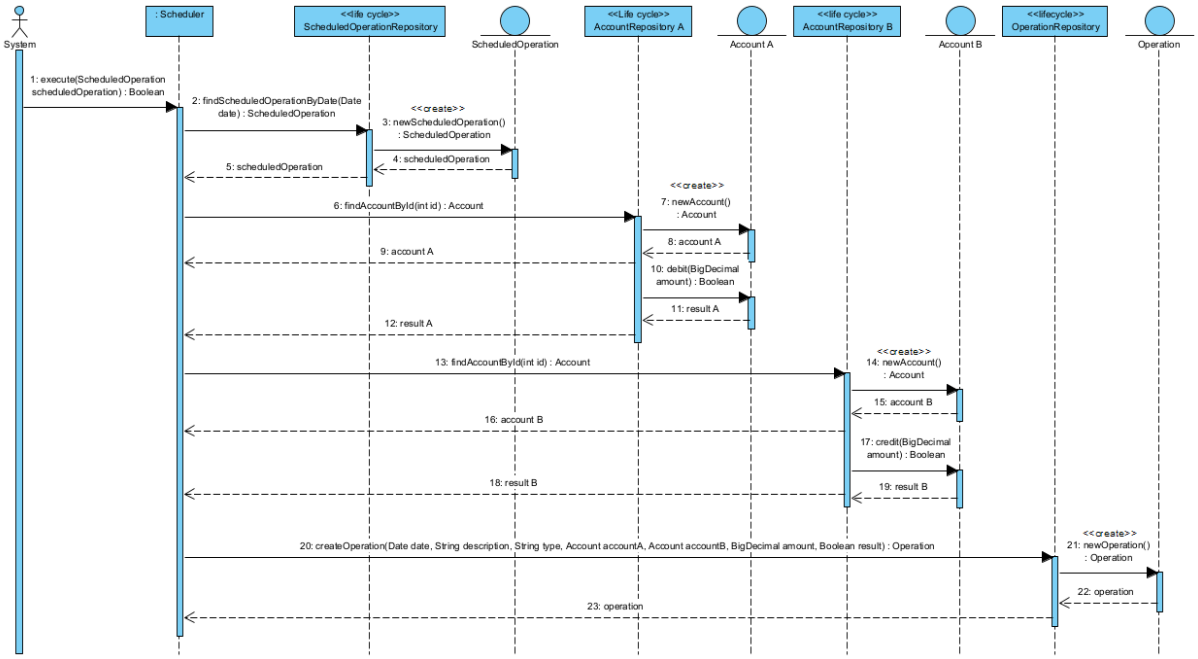

Fig. 16 : diagramme de séquences « exécuter le virement automatique »

4.2.5.2. Description des classes

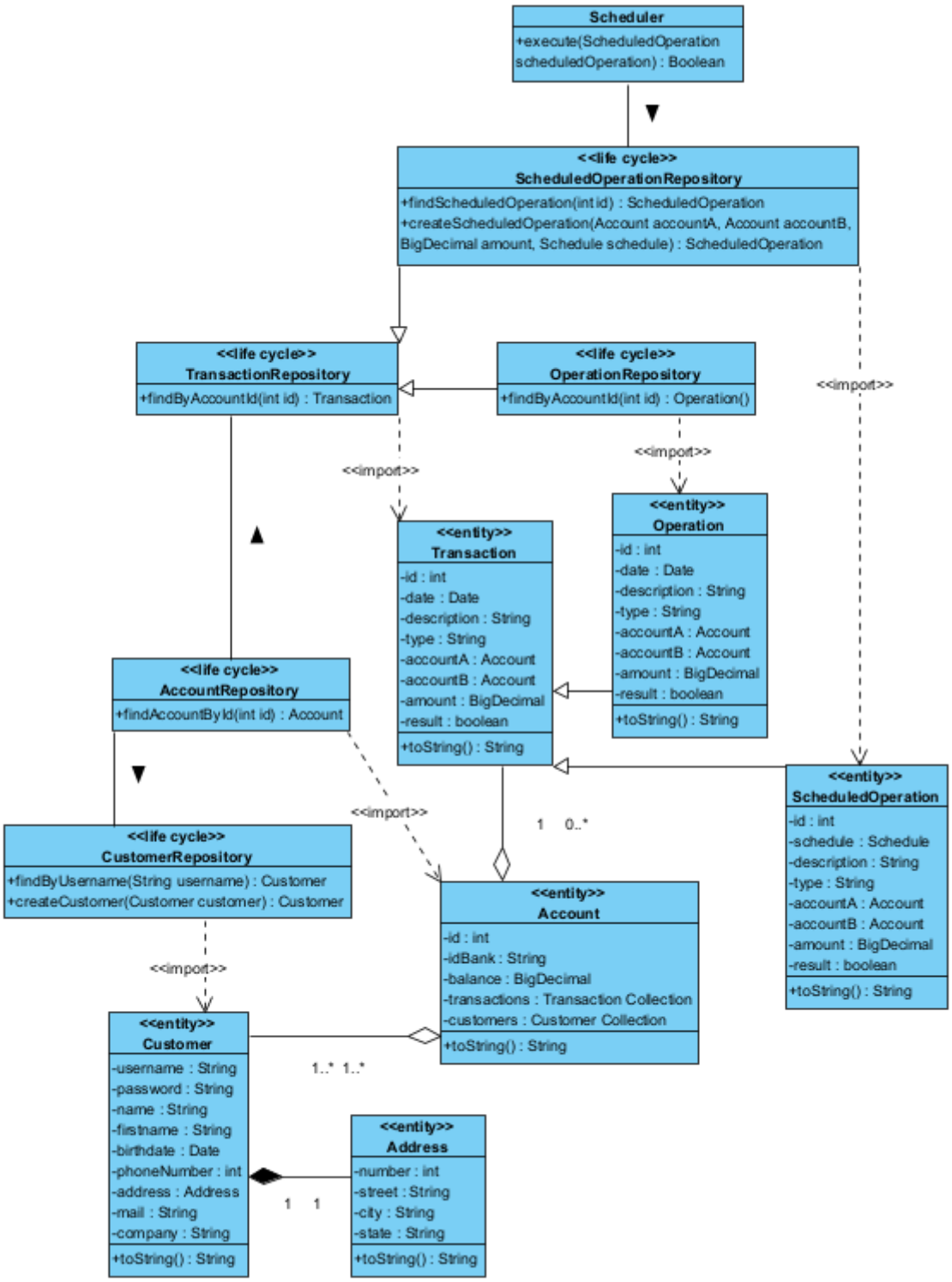

Fig. 17 : diagramme de classes « exécuter le virement automatique »

4.2.6. Recréditer le compte après le rejet du virement

Un virement peut être rejeté pour diverses raisons, par exemple :

- Le compte peut ne pas être suffisamment approvisionné pour que le montant dû puisse être prélevé. Dans ce cas, le système ne débite pas le compte émetteur du virement et affiche un message d'erreur indiquant l'insuffisance de crédit du compte.
- Le compte destinataire a été clôturé ou n'existe pas. Dans ce cas, le système ne débite pas le compte émetteur du virement et affiche un message d'erreur indiquant l'inexistence du compte destinataire.

Dans les cas suivants, le compte émetteur a été débité et suite au rejet du virement interne, le compte doit être recrédité.

- Un problème technique.
- Les données concernant le compte destinataire sont invalides et les informations techniques (nom, prénom, IBAN, etc.), transmises à la banque sont erronées.
- Le client peut demander à bloquer son compte.
- Une fraude a été détectée.
- Etc…

N.B. : Dans le cas de la mise en place d'un virement régulier, l'opération peut, par exemple, être acceptée à la première échéance, rejetée à la seconde puis de nouveau acceptée à la troisième échéance.

Ce cas d'utilisation concerne les différentes situations où le virement a été rejeté, le compte émetteur a été débité et doit donc être recrédité.
Si le compte n'a pas été débité, un message d'erreur est déclenché et l'opération est enregistrée avec le résultat d'échec de celle-ci.

4.2.6.1. Liste des objets candidats

- <<Entity>>

Customer : Objet représentant un client.
Address : Objet représentant une adresse postale.
Account : Objet représentant un compte bancaire.
Transaction : Objet représentant une opération bancaire.
Operation : Objet représentant une opération bancaire de base.

- <<Boundary>>

checkoperations.html : page affichant l'historique des opérations du compte bancaire. Suite à l'opération de recrédit du compte par le système, celle-ci sera visible dans l'historique des opérations.
AccountManagementController : servlet servant à la gestion des comptes bancaires.

- <<Control>>

AccountManagementService : Objet chargé de la logique métier de gestion de compte bancaire. Il gère l'affichage de l'objet accountManagement.html ainsi que les opérations de gestion de compte (consulter le solde, consulter l'historique des opérations, faire un dépôt, effectuer un virement interne/externe, mettre en place un virement automatique interne/externe).

- <<Life cycle>>

CustomerRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Customer persistantes, pour les rechercher par leur clé primaire et pour interroger.
AccountRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Account persistantes, pour les rechercher par leur clé primaire et pour interroger.
TransactionRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Transaction persistantes, pour les rechercher par leur clé primaire et pour interroger.
OperationRepository : interface Spring Data JPA utilisée pour créer et supprimer des instances d'entité Operation persistantes, pour les rechercher par leur clé primaire et pour interroger.

4.2.6.2. Description des interactions entre objets

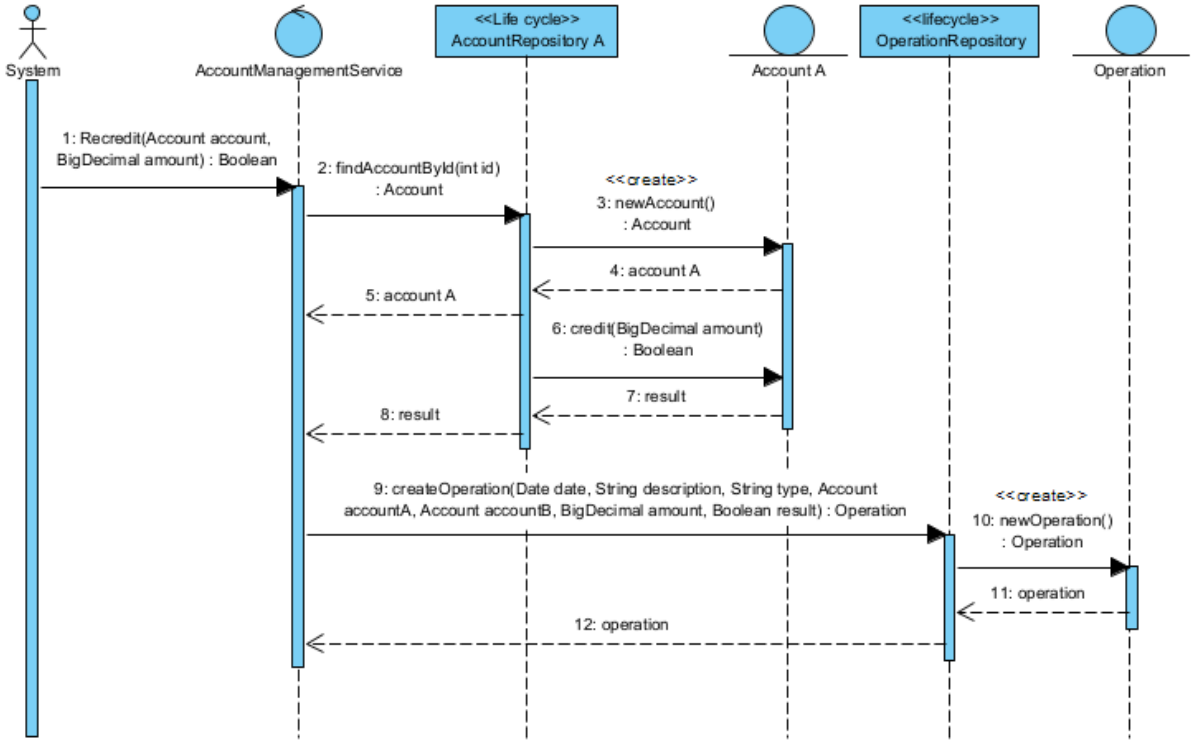

Fig. 18 : diagramme de séquences « recréditer le compte après le rejet du virement »

4.2.6.3. Description des classes

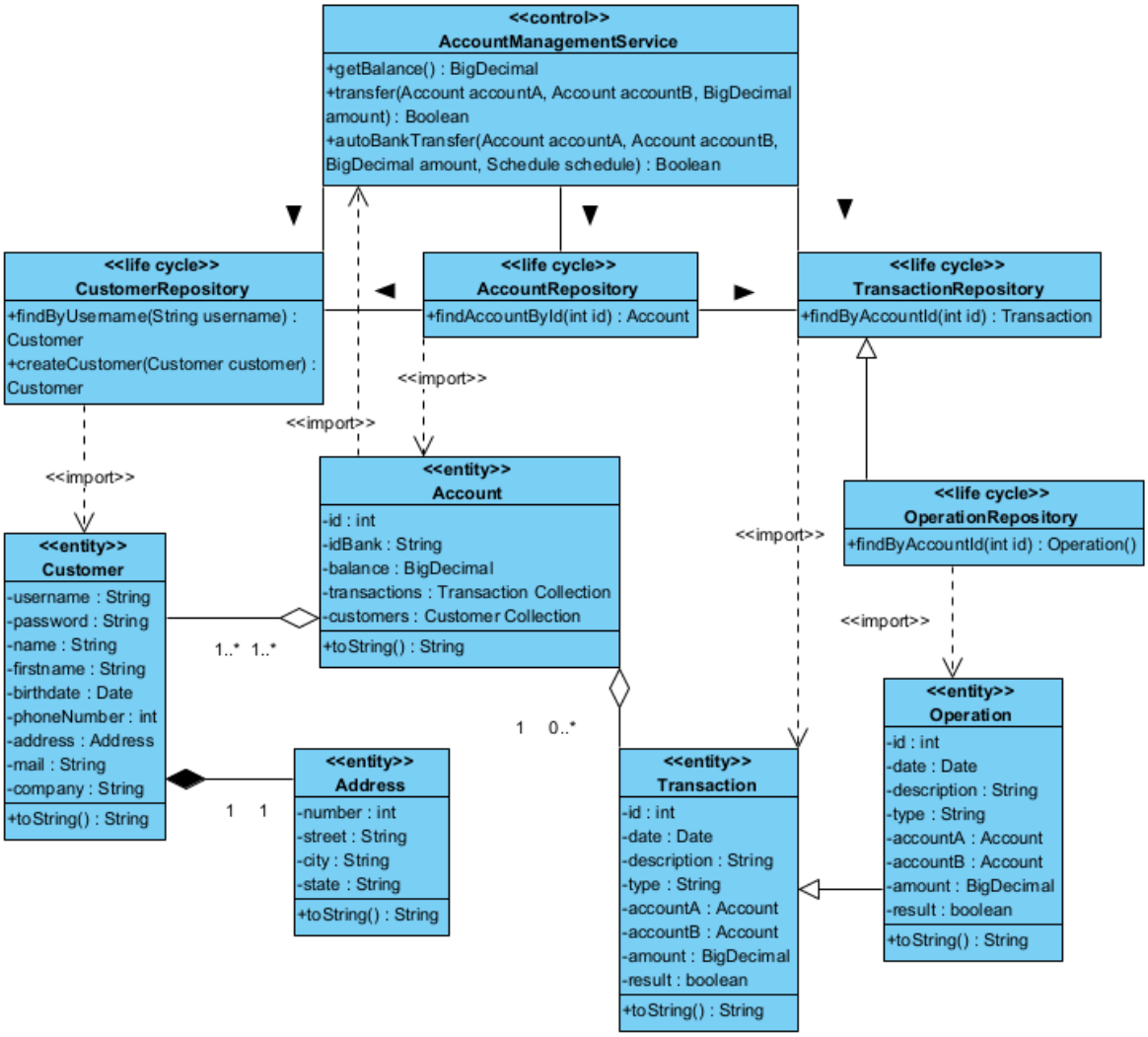

Fig. 19 : diagramme de classes « recréditer le compte après le rejet du virement »

5. Regroupement des classes

5.1. Groupe domaine

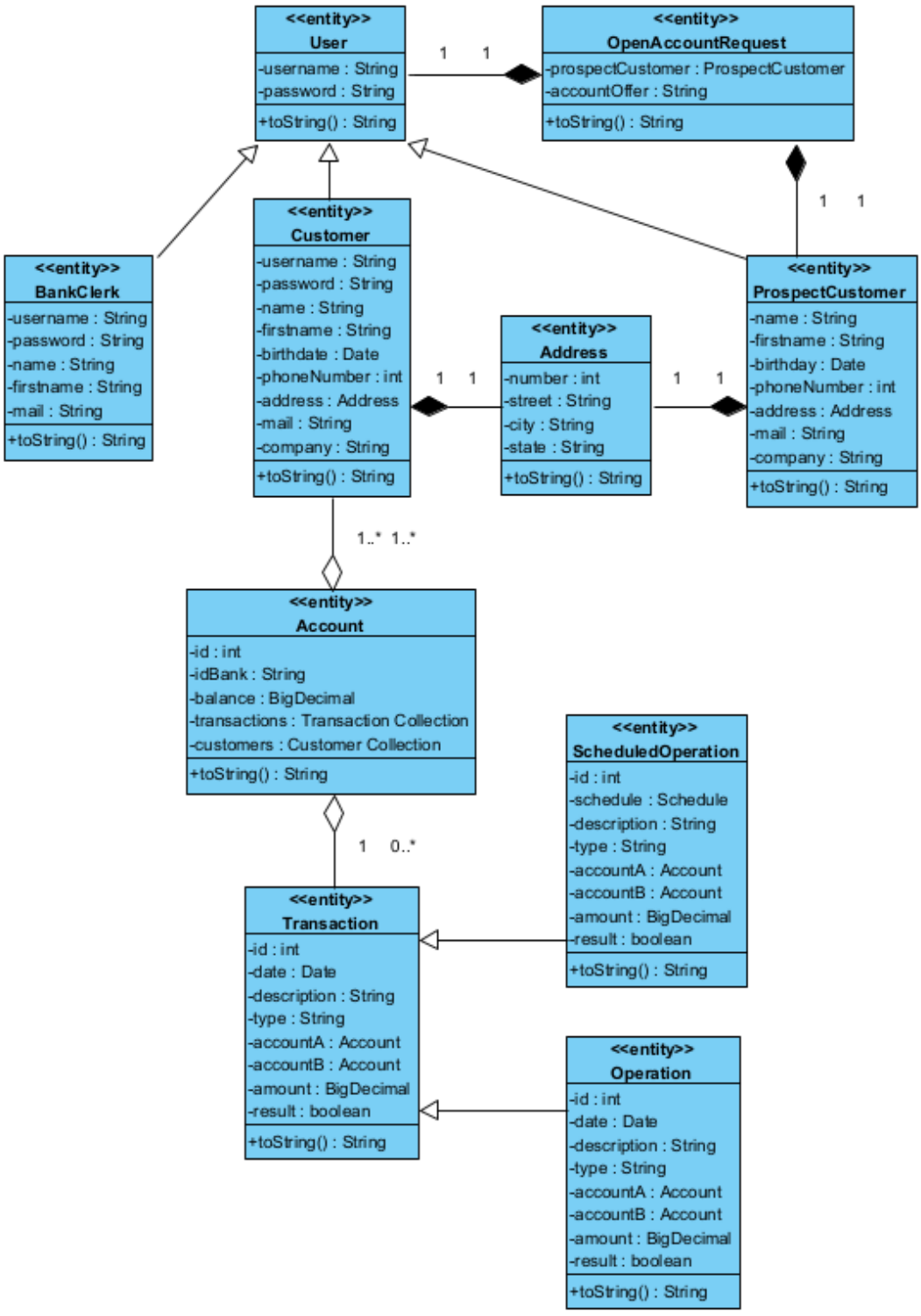

Fig. 20 : diagramme de classes « domain »

5.2. Groupe cycle de vie

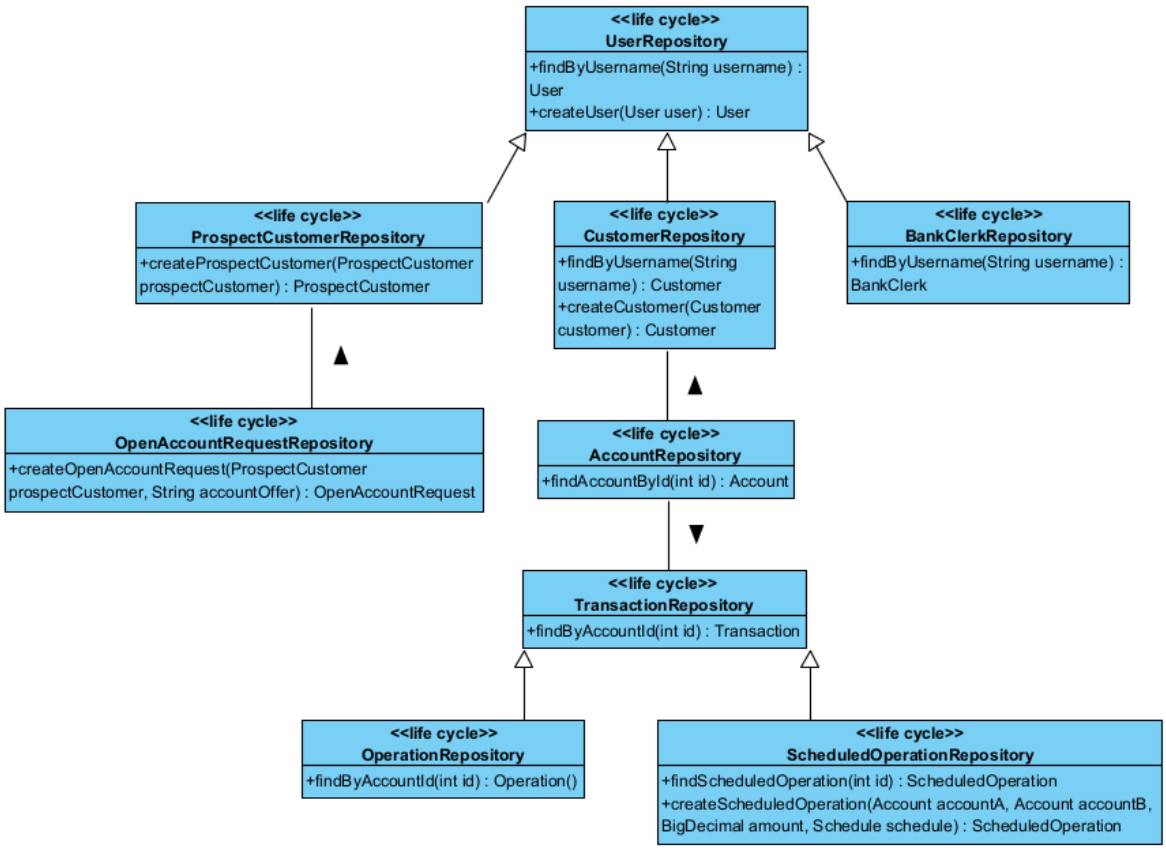

Fig. 21 : diagramme de classes « cycle de vie »

5.3. Groupe Service

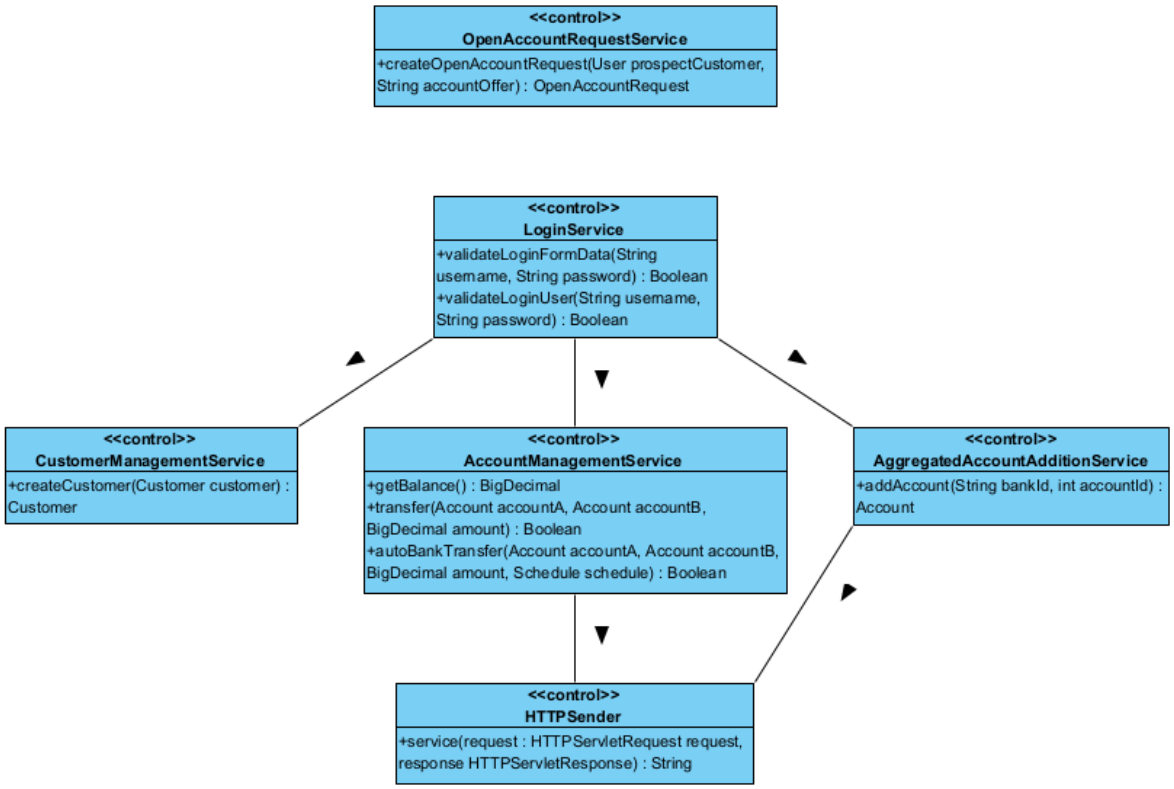

Fig. 22 : diagramme de classes « service »

5.4. Groupe interface utilisateur et système

Les interfaces utilisateur consistent en des pages Thymeleaf - Vue avec l'extension .html, aussi, l'énumération de ces pages ne paraît pas pertinente. Les classes @Controller ci-dessous sont chargées de préparer une carte de modèles avec des données et de sélectionner une vue à afficher, c'est-à-dire les pages Thymeleaf – Vue avec les données métier.

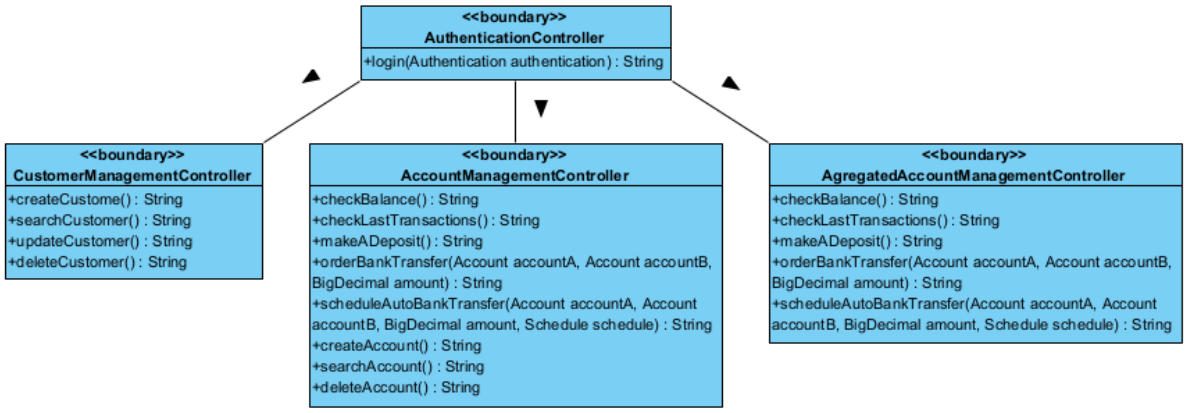

Fig. 23 : diagramme de classes « interface utilisateur et système »

6. Annexes

6.1. Terminologie

Balance	Solde bancaire.
Bank clerk	Employé (authentifié) de Mocha Bank.
Bank transfer	Virement bancaire.
Boundary	Les objets « boundary » se trouvent à la frontière entre un système ou un sous-système et un acteur. Il s'agit des interfaces système.
Control	Les objets « control » assurent une coordination d'autres objets. Il s'agit d'une façade entre objet « boundary » et objet « entity ».
CRUD	Create, Read, Update, Delete (Créer, Lire, Mettre à jour, Supprimer).
Customer	Client authentifié de Mocha Bank.
Deposit	Dépôt.
Entity	Les objets « entity » encapsulent les données métier et la logique métier du système [Jacobson, et al. 1999]. (informations durables et persistantes)
Lifecycle	Les objets « life cycle » assurent le suivi des objets « entity ». Les fonctions communes d'un objet « life cycle » incluent la création, la destruction et la localisation des objets « entity ».
Mock	Objet simulé reproduisant le comportement d'un objet réel de manière contrôlée.
Screen scraping	Technique permettant d'accéder aux données d'un client d'une banque en utilisant ses codes d'accès.
Transaction	Opération bancaire.
User	Utilisateur de l'application Mocha Bank.

7. Index des diagrammes

8. Sources

- Cours GLG203 - Architecture logicielle Java 1 – CNAM Paris
- https://spring.io
- http://tomcat.apache.org
- https://www.thymeleaf.org
- https://vuejs.org
- https://www.mysql.com
- http://httpunit.sourceforge.net
- https://greenmail-mail-test.github.io
- https://www.journaldunet.com/economie/finance/1423134-api-dsp2-ou-en-sont-les-banques-francaises
- Veena K.Katankar et. al. / (IJCSE) International Journal on Computer Science and Engineering
- Patterns of Enterprise Application Architecture par Martin Fowler