INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

DOSSIER DE CONCEPTION

Génie logiciel – INSA Lyon 2017

Objectif

Ce dossier a pour but de définir la conception de notre application. Nous allons donc présenter l'architecture globale du système que nous proposons, les différents logiciels qui seront développées, le matériel utilisé et le matériel conçu spécifiquement pour ce projet.

Auteurs

Marc-Antoine Fernandes
Julia Lu Dac
Lucas Ono
Tianhao Wu
Ye Yuan

Version 2
Date de création : 05/04/17
Dernière modification : 20/05/17

Table des matières

1 Organisation générale du système

Le système sera composé de plusieurs parties distinctes :

- Des serveurs (comportant des dictionnaires)
- Des applications (terminaux) clients des laboratoires

Un client est un laboratoire qui dispose de fichiers contenant les analyses médicales et d'un fichier contenant les adresses des serveurs à interroger. Chaque terminal d'un client enverra une requête contenant l'analyse (fichier du génome) par web vers un ou plusieurs serveurs de son choix. En réponse, il recevra :

- une liste des risques de maladies associée à l'analyse envoyée s'il désire une évaluation complète.
- une réponse oui/non s'il décide de connaître la présence d'une maladie dans une analyse (évaluation spécifique)

Chaque serveur possède son propre dictionnaire qui contient une liste de maladies et les mots associés. Le serveur va faire l'analyse lorsqu'il reçoit une requête :

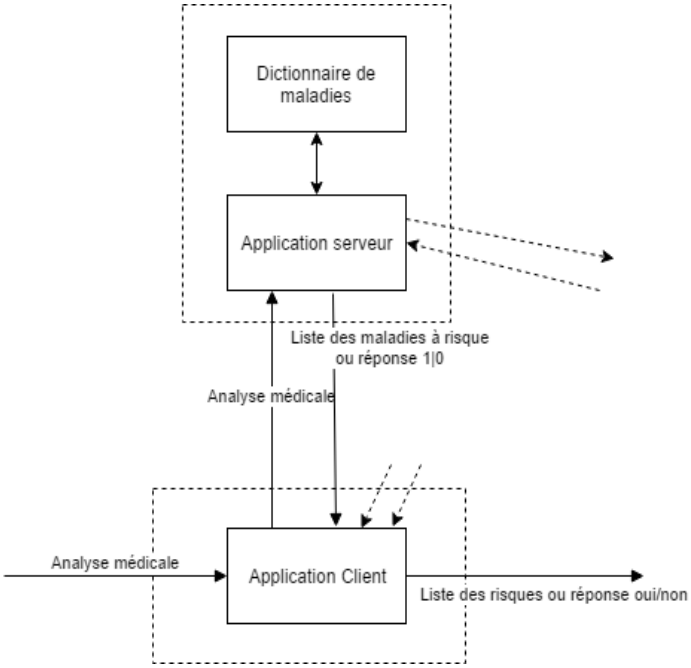
- pour une évaluation complète, si tous les mots associés à une maladie sont présents, alors cette maladie est mise dans la liste de maladies à renvoyer au client.
- pour une évaluation spécifique, si tous les mots associés à la maladie cherchée présents, alors la réponse sera oui (1), autrement elle sera non ().

Lorsque le serveur a fini l'évaluation, il va renvoyer, suivant le cas, la liste de maladies ou la réponse au client.

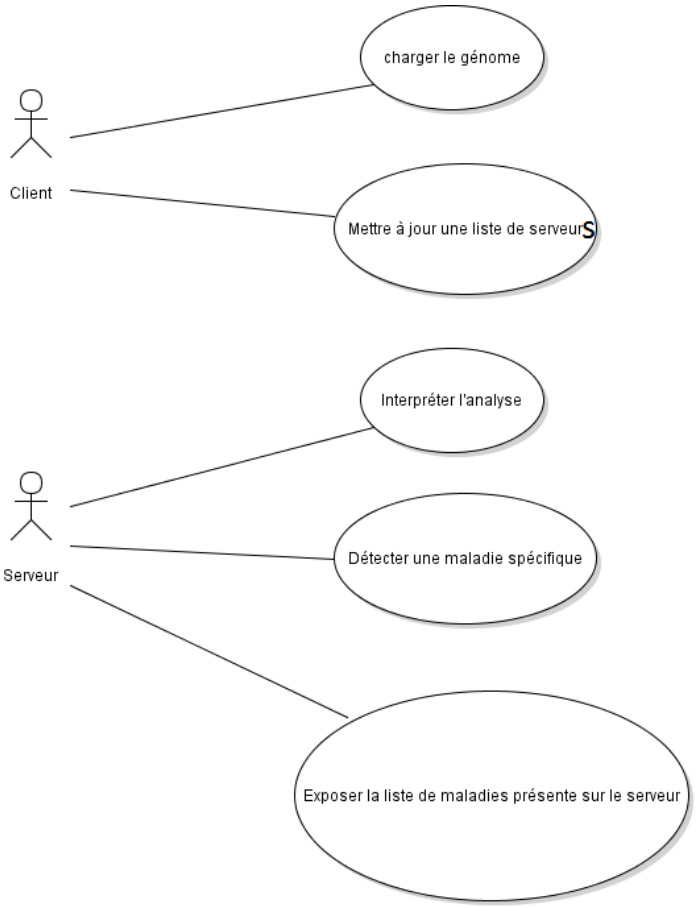
2 Architecture de l'application

2.1 Schéma général

Voici un schéma montrant brièvement les échanges (impliquant l'analyse médicale) entre clients et un serveur.



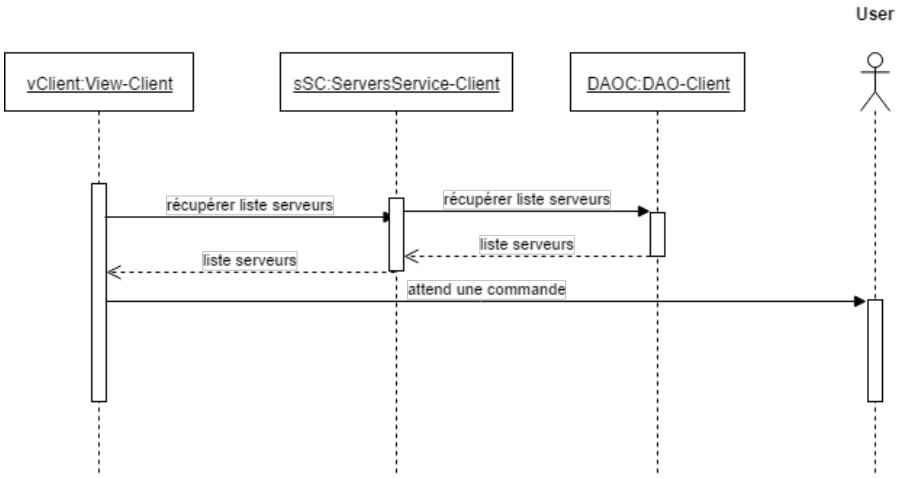
2.2 Diagramme de cas d'utilisation



2.3 Déroulement de l'application

2.3.1 Diagrammes de séquence

2.3.1.1 Démarrage de l'application cliente

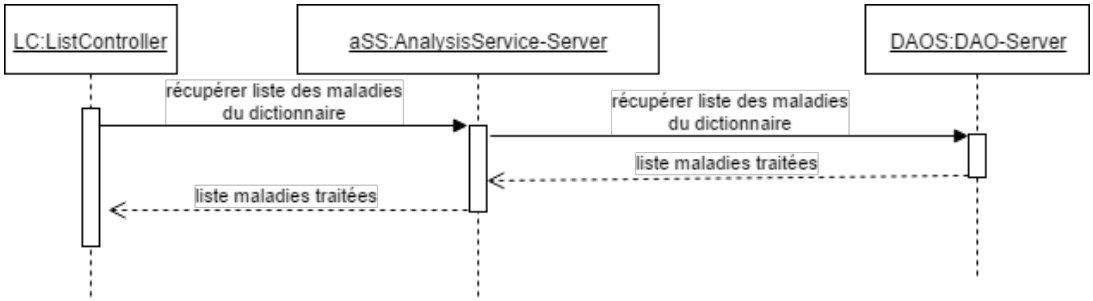


A son démarrage, l'application charge la liste des serveurs stockée dans le fichier sur le disque dur.

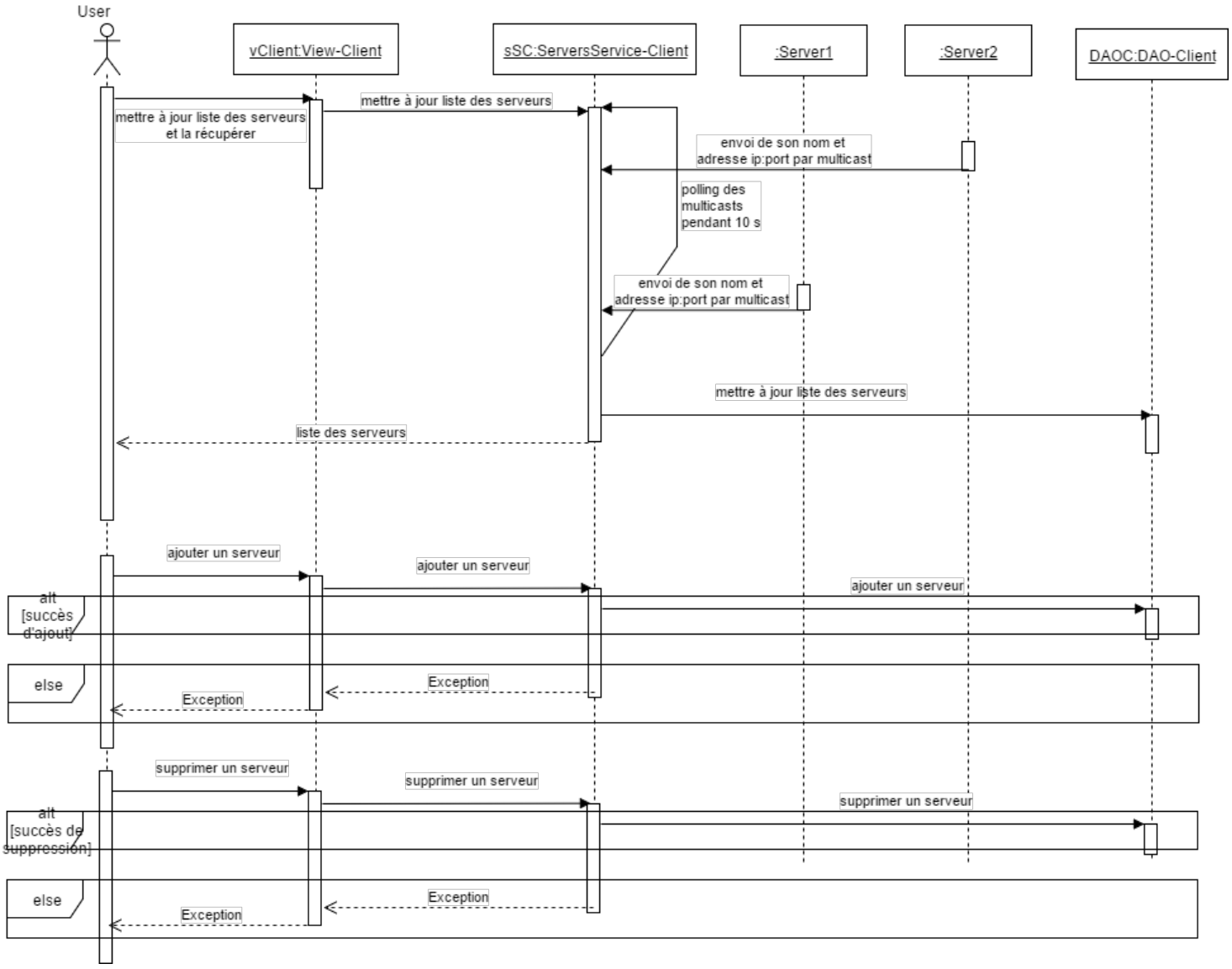
Après le démarrage, le client peut entrer des commandes. Plus tard, il pourra à tout moment mettre à jour la liste des serveurs (voir diagramme 2.2.1.3 Gestion de la liste des serveurs)

2.3.1.2 Démarrage de l'application serveur

Le serveur charge le dictionnaire contenant toutes les maladies de son fichier



2.3.1.3 Gestion de la liste des serveurs (côté client)



2.3.1.4 Demande d'une évaluation complète

Le diagramme étant trop volumineux, nous avons préféré séparer en trois parties

Diagramme partie 1 : représentation des interactions entre User, View et AnalysisService

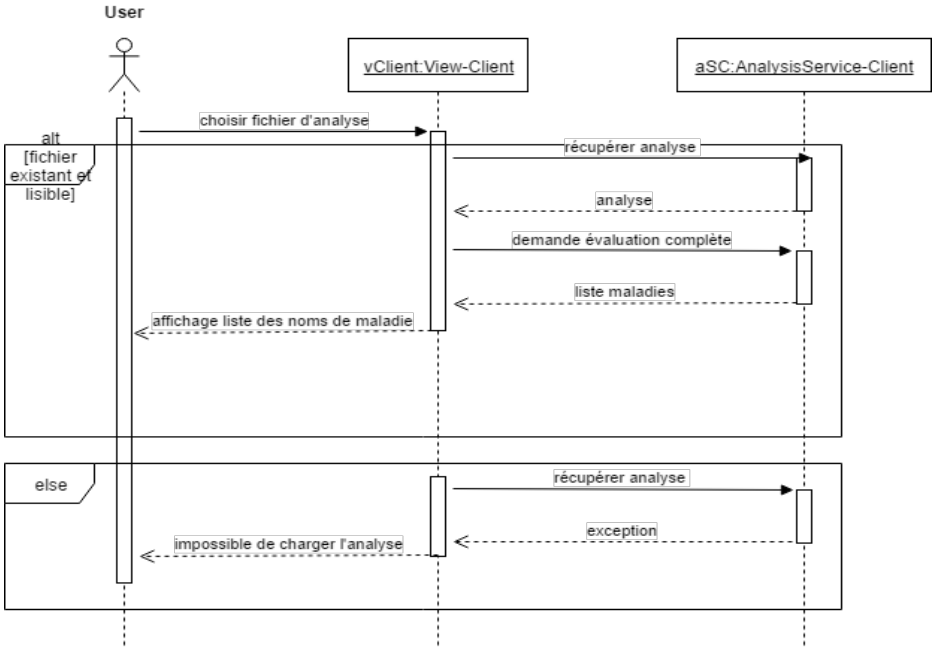


Diagramme partie 2 : représentation des interactions entre AnalysisService, DAO et les serveurs

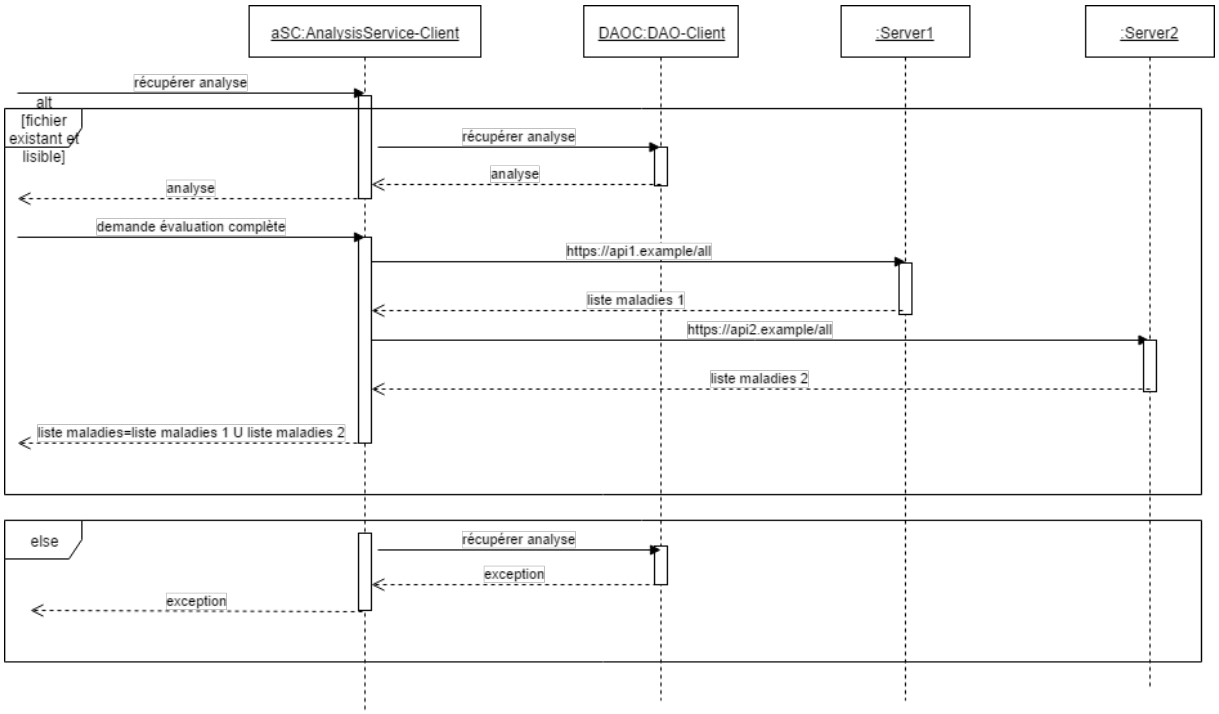
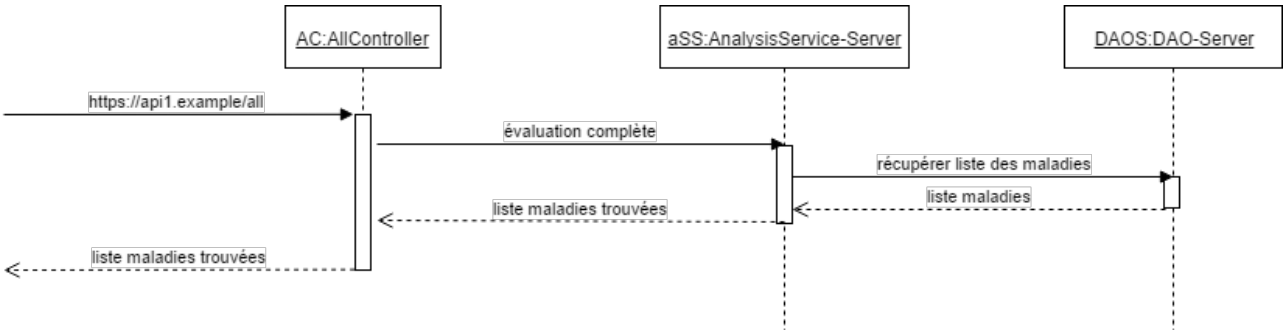


Diagramme partie 3 : représentation des interactions entre modules d'un serveur



2.3.1.5 Demande d'une évaluation spécifique (vue côté client)

Le diagramme étant trop volumineux, nous avons préféré séparer en trois parties

Diagramme partie 1 : représentation des interactions entre User, View et AnalysisService

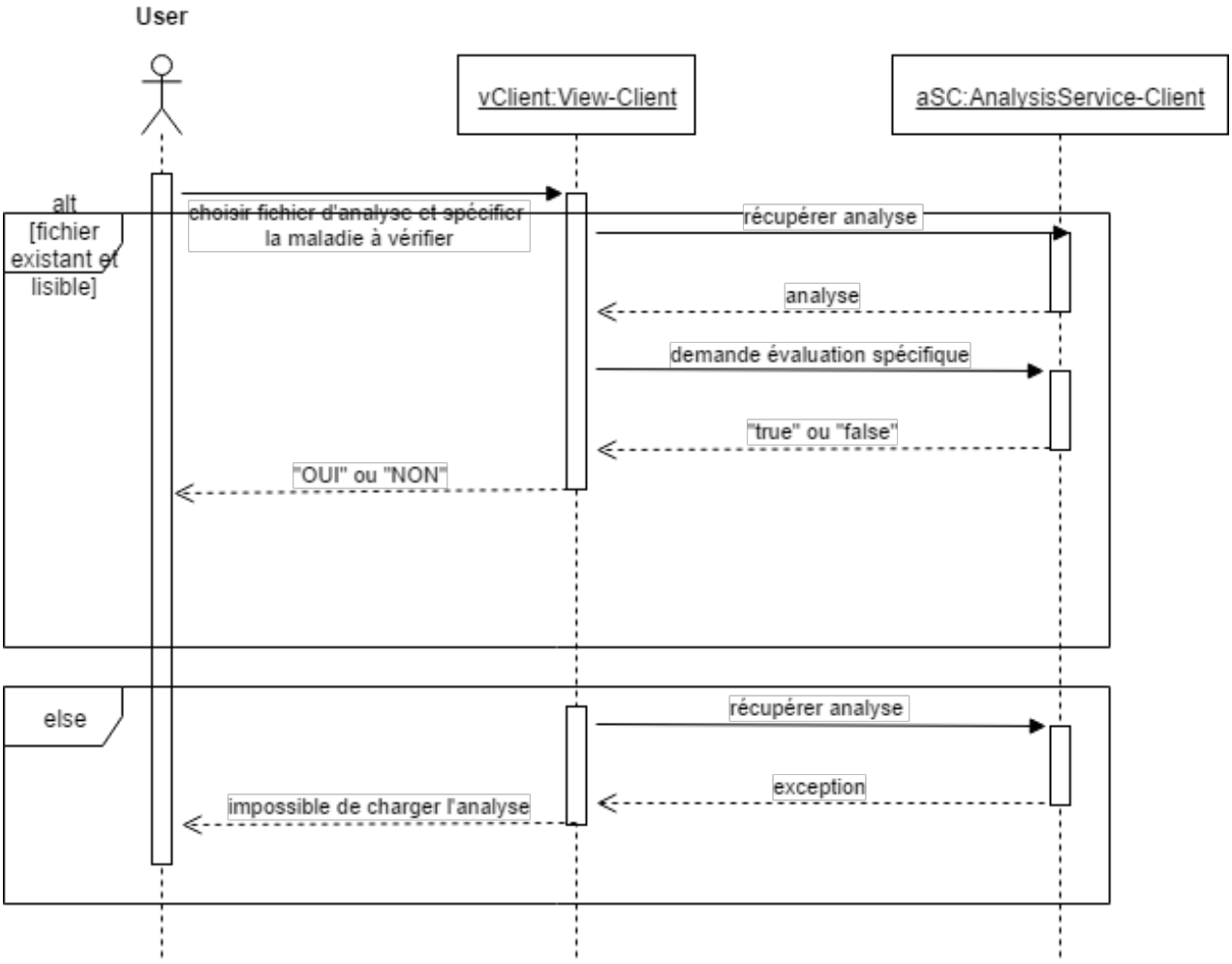


Diagramme partie 2 : représentation des interactions entre AnalysisService, DAO et les serveurs

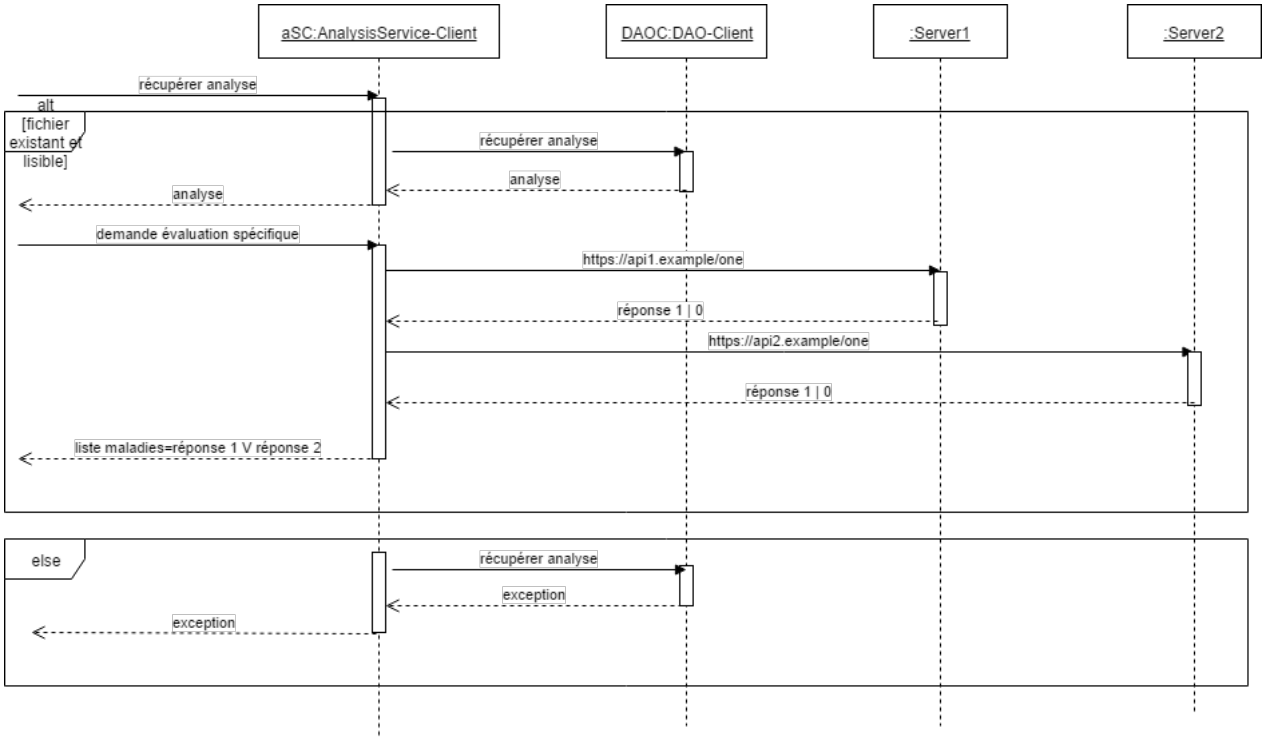
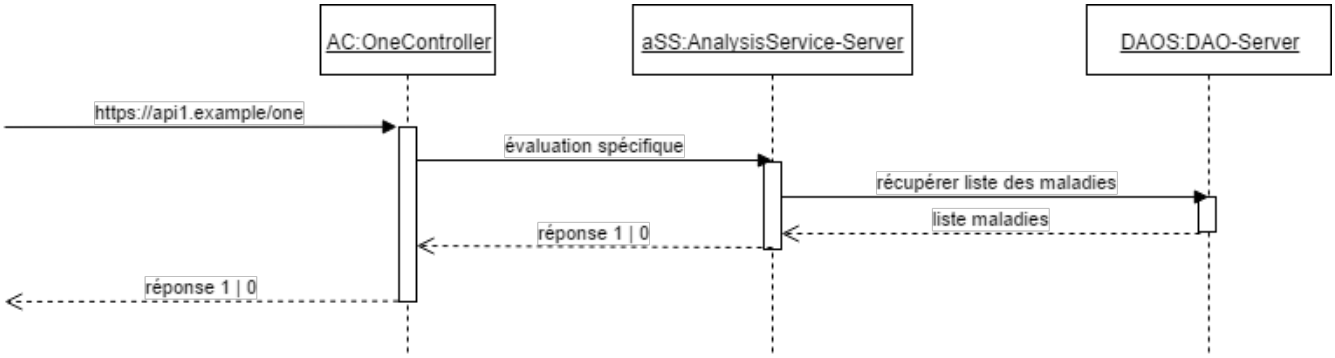
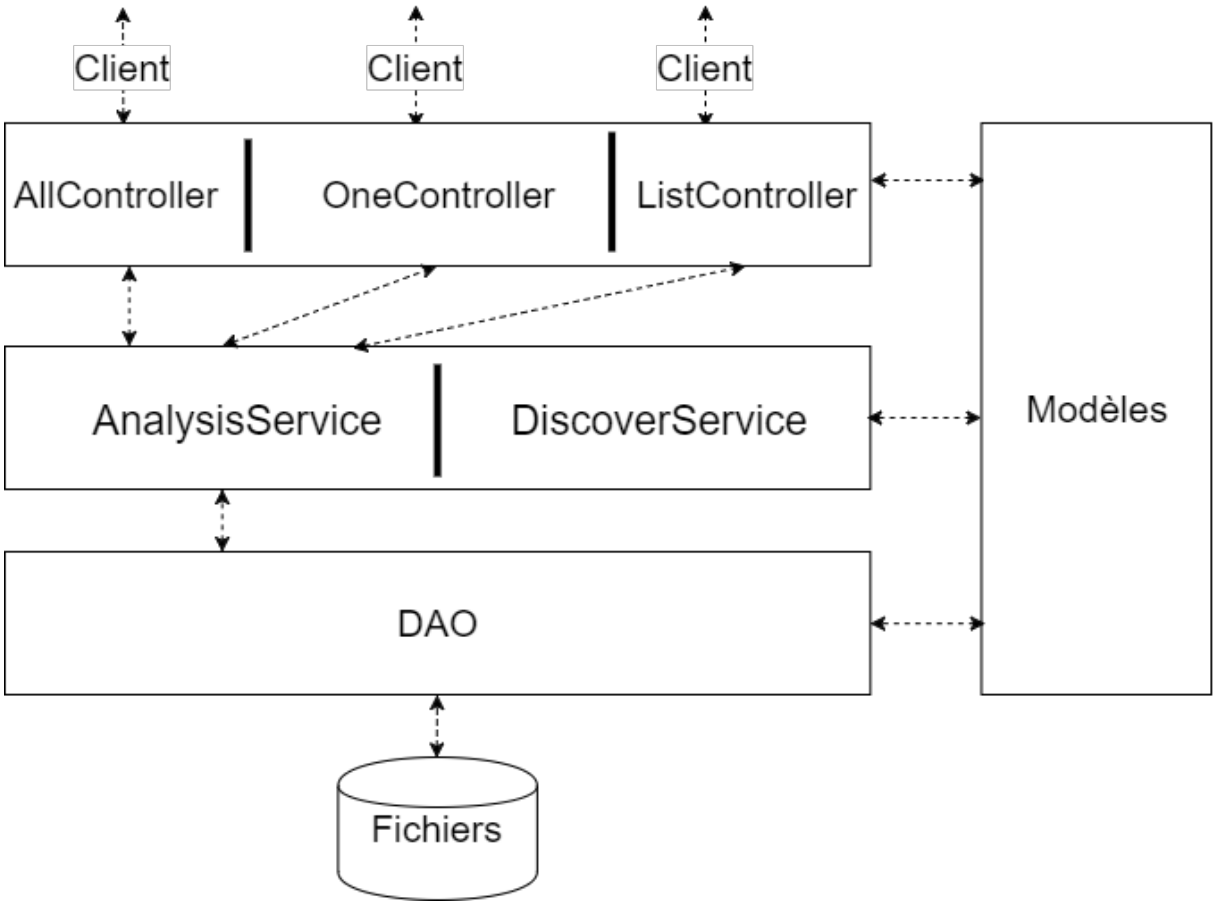


Diagramme partie 3 : représentation des interactions entre modules d'un serveur



2.4 Architecture application serveur

Le serveur sera découpé en plusieurs modules en suivant une architecture multicouche approchant un modèle MVC :



2.4.1 Comme on peut le voir ci-dessus, l'application sera découpée en 4 modules :

Les modèles

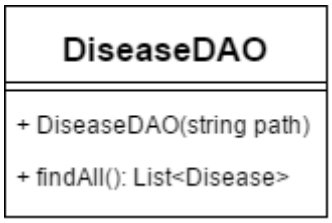
Cette partie contient les différents modèles de données. Ces modèles circulent dans toute l'application. Il y aura deux objets :

- **Genome** : modélise une analyse médicale, il est défini par une liste de mots.
- **Disease** : modélise une maladie, défini par un nom et une liste de mots qui sont associés au risque de cette maladie.

Un mot est représenté par une chaine de caractères composée de ces 4 caractères possibles : A, T, C, G.

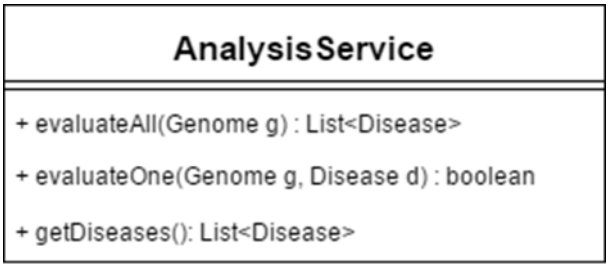
2.4.2 Les DAO

Cette partie contient l'ensemble des classes qui permettront de charger dans la mémoire les maladies en tant qu'objets. Il y aura un seul objet DAO, l'objet de chargement des maladies depuis un fichier. Ce chargement sera effectué au lancement de l'application.



2.4.3 Les Services

Les services sont le cœur de l'application. Ici, l'application aura un service qui évaluera un génome afin d'en sortir un résultat ou enverra la liste des maladies traitées (AnalysisService) et un service qui enverra à intervalles réguliers son adresse IP et le port de connexion sur une adresse multicast définie dans l'application (DiscoverService).



Les maladies fonctionnent par système de clé/valeur, pour une maladie, une liste de mots est associée. A noter qu'une maladie peut être définie de plusieurs façons dans un même dictionnaire (il peut y avoir plusieurs clés pour une même valeur).

Si le génome contient l'intégralité de la liste de mots de la maladie (de façon pas forcément ordonnée ni séquencée), la maladie associée doit être ajoutée à la liste des maladies détectées pour cette analyse.

Afin que des clients puissent détecter le serveur, celui-ci va utiliser le système multicast pour partager son adresse IP ainsi que son port de connexion. L'intervalle sera de 2.5 secondes. Un service dédié permettra de démarrer et arrêter la diffusion.

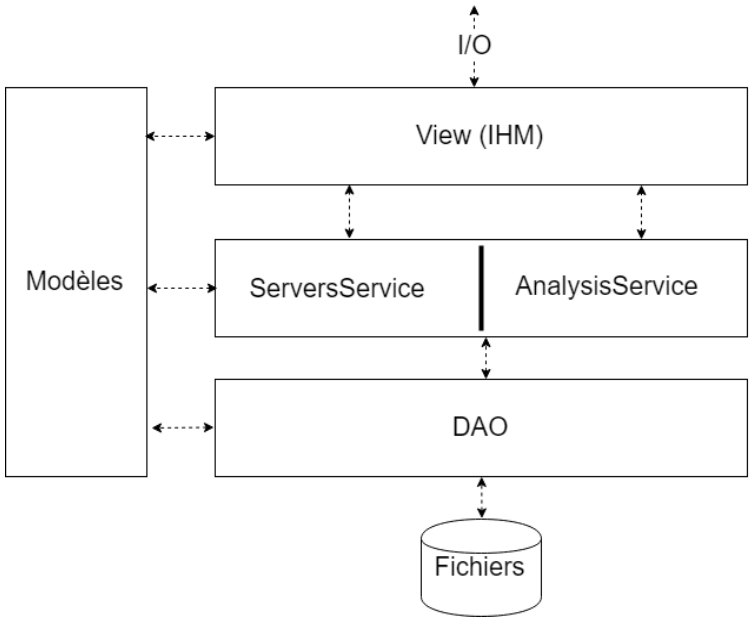
2.4.4 Les Controllers

Les controllers permettent aux clients de faire des requêtes au serveur. Il y en a 3 :

- **AllController** : celui-ci a pour but de traiter les requêtes d'évaluation complète (détection de toutes les maladies présentes sur le serveur).
- **OneController** : celui-là a pour but de traiter les requêtes d'évaluation d'une seule maladie (renvoi vrai ou faux).
- **ListController** : ce controller permet aux clients d'avoir la liste courante des maladies traitées par le serveur.

2.5 Architecture application client

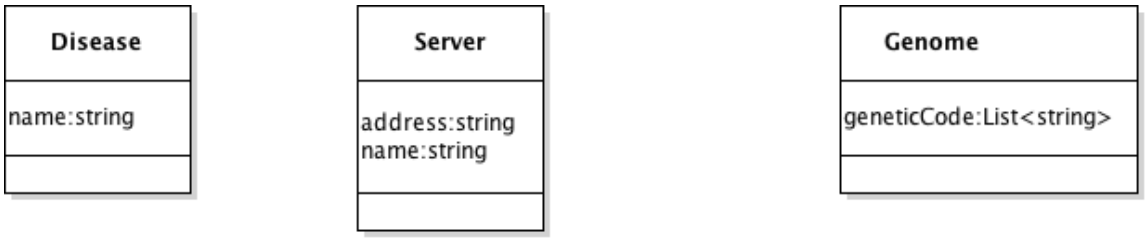
Le client sera également découpé en plusieurs modules en suivant une architecture multicouche approchant un modèle MVC :



2.5.1 Modèles

Cette partie contient les différents modèles de données. Ces modèles circulent dans toute l'application. Il y en a 3 :

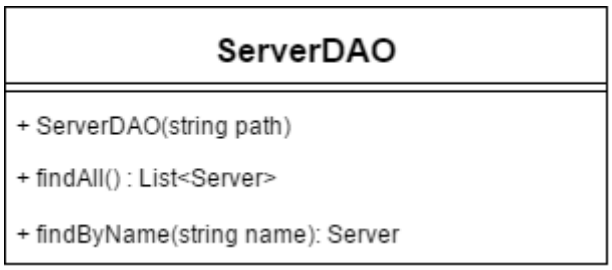
- **Genome** : modélise une analyse médicale, il est défini par une liste de mots.
- **Disease** : modélise une maladie, définie par un nom.
- **Server** : modélise un serveur, défini par une adresse de connexion (port inclus) et un nom d'affichage.



Les DAO

Cette partie contient l'ensemble des classes qui permettront de charger dans la mémoire les serveurs en tant qu'objets. Il y aura un seul objet DAO, l'objet de chargement des serveurs depuis un fichier. Ce chargement sera effectué au lancement de l'application et le fichier sera mis à jour à chaque changement de données.

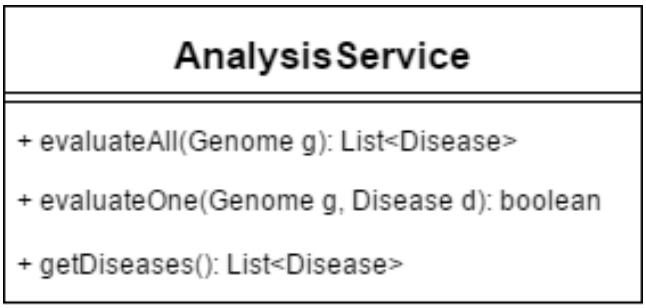
2.5.2



Les Services

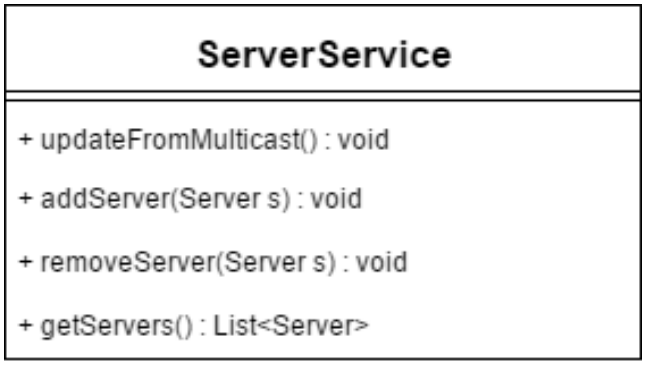
2.5.3

Les services sont le cœur de l'application. Ici, l'application aura un service qui fera les demandes aux différents serveurs pour une analyse (AnalysisService) et un autre qui permettra la gestion de la liste des serveurs (ServerService).



Lorsqu'une évaluation d'une analyse pour une maladie spécifique est faite, le résultat est vrai si au moins un serveur renvoie « 1 », faux sinon.

Le service peut également récupérer une liste de maladies. Celle-ci peut servir à aider l'utilisateur à sélectionner une maladie.



La mise à jour de la liste des serveurs depuis Multicast rajoute à la liste les serveurs détectés. Si ceux-ci sont déjà présents dans la liste (à travers l'adresse IP et port) alors on ne les rajoute pas (pour éviter les doublons). L'application écoutera pendant 10 secondes l'adresse multicast.

La vue

L'IHM sera faite avec MFC de Visual Studio.

2.5.4

3 Réflexion sur les données

3.1 Format de données

- Format d'une analyse médicale
    - Fichier texte
    - Chaque mot = chaîne de caractères (4 caractères possibles : A, T, C, G)
    - Un mot par ligne
- Format d'un dictionnaire
    - Fichier texte
    - Une signature de maladie par ligne
    - Chaque ligne composée de mots séparés par le symbole ';'
    - Premier mot : le nom de la maladie – tout caractère ASCII non-blanc et différent de ';'
    - Le reste de mots : caractérisation de la maladie ; fin de ligne : '\r\n'
- Format de message
    - Chaîne de caractères terminée avec le caractère '\0' ; plusieurs mots (trois) dans la chaîne séparés avec ';'
    - Premier mot : version du protocole : 'MA v1.0'
    - Deuxième mot : adresse IP du serveur
    - Troisième : port d'écoute
- Format de demande
    - Demande de la liste de maladies qu'un serveur peut analyser
        - Format : 'MA v1.0\r\nGET DISEASES\r\n\r\n'
        - Réponse : 'MA v1.0\r\nDESEASES\r\n<D1>\r\n…<Dn>\r\n\r\n'
    - Demande d'analyse pour une AM et toutes les maladies
        - Format : 'MA v1.0\r\nCHECK ALL\r\n\r\n'
        - Réponse : 'MA v1.0\r\nDESEASE <D1>\r\n…DESEASE <Dn>\r\n\r\n'
    - Demande d'analyse pour une AM et une maladie précise
        - Format : 'MA v1.0\r\nCHECK DISEASE\r\n<D>\r\n\r\n'
        - Réponse : 'MA v1.0\r\nDESEASE <D>\r\n<0|1>\r\n\r\n'

3.2 Protocole de communication

- Découverte via multicast
- Adresse : 225.6.7.8
- Port : 5678

3.3 Volumétrie

Il est nécessaire d'avoir une idée précise du volume que représente toutes les données afin de pouvoir dimensionner le système.

Une analyse médicale est composée de moins de 1M de mots et une maladie de 10% de ces 1M de mots (les maladies sont stockées dans les serveurs).

4 Gestion des problèmes, anomalies et sécurité

4.1 Gestion des exceptions

Côté serveur

- **AnalysisException** : Lancée lorsque le serveur n'arrive pas à évaluer le génome (mauvais format, …). Cette exception n'est pas critique pour le serveur et sera juste retransmise au client.

4.1.1

- **DiseaseNotFoundException** : Lancée lorsque la maladie envoyée par le client n'existe pas sur le serveur. Cette exception n'est pas critique.

- **ReadException** : Lancée lorsque le serveur n'arrive pas à lire le dictionnaire des maladies. Cette exception est critique et conduira à l'arrêt du serveur (avec un message d'erreur).

Côté client

4.1.2

- **ReadException** : Lancée lorsque le client n'arrive pas à lire un génome (inaccessible, mauvais format). Cette exception n'est pas critique et devra être retransmise à l'utilisateur.

- **ServiceException** : Lancée lorsqu'une erreur s'est produite dans un service (impossible d'enregistrer les serveurs, etc…). Cette exception n'est pas critique.

- **CommunicationException** : Lancée lorsqu'il y a un problème de communication avec un serveur. Cette exception n'est pas critique et peut être interprétée de plusieurs façon (cf. 4.2 Problème de communication client-serveur).

N.B. : Les exceptions côté serveur qui sont retransmises au client seront affichées à l'utilisateur.

4.2 Problème de communication client-serveur

Lors de la communication client-serveur, une erreur peut survenir (erreur réseau, serveur indisponible, …). Dans ces cas-là, l'application client doit notifier l'utilisateur d'une erreur réseau.

Si l'application client n'a pas réussi à contacter au moins un serveur, celle-ci indiquera à l'utilisateur une probable panne de sa connexion au réseau. Si certains serveurs sont indisponibles, l'application indiquera juste à l'utilisateur que certains serveurs ne sont pas disponibles.