Université des Sciences de Montpellier
Master 2 Semestre 1
Unité d'Enseignement FMIN306

Raja - Bases de données distribuées

16 décembre 2010

Audrey Novak
Romain Maneschi
Jonathan Fhal
Aloys Urbain

Table des matières

Table des figures

1 Introduction

1.1 Projet

Ce projet permettra d'accéder à différentes informations concernant les maladies dans le monde. Ceci de la façon la plus simple possible. Dans un premier temps nous devrons interroger le système par le biais de requêtes sql simplifiées. Mais à terme une interface permettrait de simplifier la construction de ces requêtes.

Les données sur les maladies sont stockées sur différentes bases de données. Ces bases de données sont stockées sur des SGBD hétérogènes non seulement par leur implantations mais aussi et surtout par leur langage de communication. Toute la difficulté du projet sera de rassembler toutes les données, pour offrir à l'utilisateur une vue d'ensemble sur ce qu'il cherche.

1.2 Equipe

Pour réaliser ce projet nous sommes une équipe de quatre étudiants en master deuxième année à la faculté des sciences de Montpellier. Ce projet intervient à l'issu de l'unité d'enseignement "Gestion de données distribuées - Intégration - Médiation".

1.3 Organisation

Tout le monde à participer à une réflexion préliminaire pendant les deux premières semaines. Ayant chacun nos propres affinitées l'organisation du travail c'est fait tout naturellement par la suite.

- Romain Maneschi : chef de projet, rédaction du rapport, organisation générale
- Audrey Novak : modèles ontologiques et traduction D2RQ par spécification N3
- Aloys Urbain : installation d'un environnement de test
- Jonathan Fhal : création des scripts d'installation et de tests des bases de données

Cette organisation nous permet de pouvoir tous commencer en même temps. Puis au fur et à mesure que chacun aura fini sa partie, nous nous rejoindrons sur le développement des classes Java pour réaliser le serveur.

1.4 Matériel

Afin de simplifier nos échanges nous avons ouvert un site internet sur google code. Ce site nous permet d'avoir un gestionnaire de versions centralisé, un wiki et un compte ftp. Tout ceci nous permet donc de réunir en un seul endroit tous nos travaux.

Pour ne pas perdre de temps, un seul membre du groupe, sera en charge de créer une machine virtuelle sur laquelle tournera les différentes bases de données. Cette machine virtuelle pourra enssuite être transmise aux autres membres, ceux-ci auront donc directement un système opérationnel.

Les SGBD utilisés seront Oracle, MySQL et PostgreSQL. Nous les avons choisis car ce

sont trois logiciels qui ont fait leur preuves et sont utilisés tous les jours par différents professionnels.

2 Modèle d'analyse

2.1 Besoins

Notre serveur offrira à un utilisateur la possibilité de chercher des renseignements sur des maladies. Plus exactement il pourra demander au système des informations sur une maladie : son nom, son facteur de transmission, le virus associé à cette maladie et un nombre de cas rencensés pour une zone (voir le schéma des tables des bases de données pour de plus amples informations). Pour cela nous avons mis en place un système de requêtes basées sur sql. Ces informations seront récupérées par nos soins auprès de l'OMS.

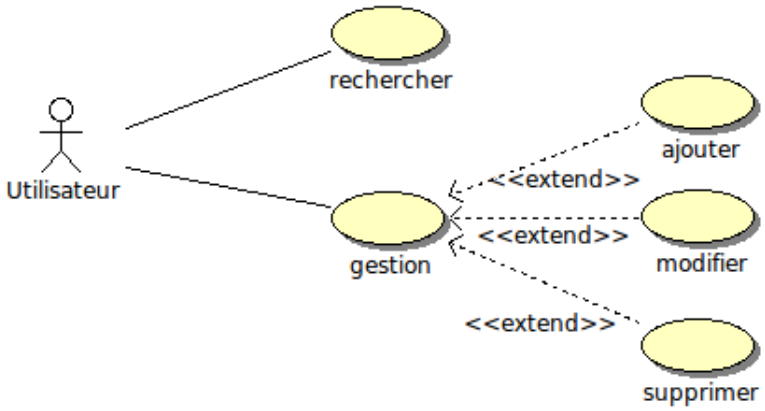

FIGURE 1 – Diagramme de cas d'utilisation

2.2 Requêtes

Notre serveur prendra en entrée une requête dont la syntaxe pourra correspondre soit au langage SQL soit au langage SparQL. SparQl sera réservé aux requêtes pour la sélection alors que SQL commandera l'insertion, la modification et la suppression. Mais notre système ne pourra gérer toute la complexité de ces langages. Nous le limiterons donc à quelques instructions : select, insert, update, delete.
De plus, et toujours dans un soucis d'efficacité, nous n'accepterons dans un premier temps, que des instructions dites basiques (toutes les instructions entre crochets sont optionnelles) :

- SELECT ?variable [, ?variable2...] WHERE ?variable_ou_valeur relation ?variable_ou_valeur
- insert into tabel_name values(value_table, value_table2...) [where column_name = value and column_name2 = value2...] ;
- update table_name set column_name = value1 [, column_name2 = value2] [where column_name = value...] ;
- delete from table_name [, table_name2] [where column_name = value...] ;

Nous avons donc mis en place un type requête particulier :

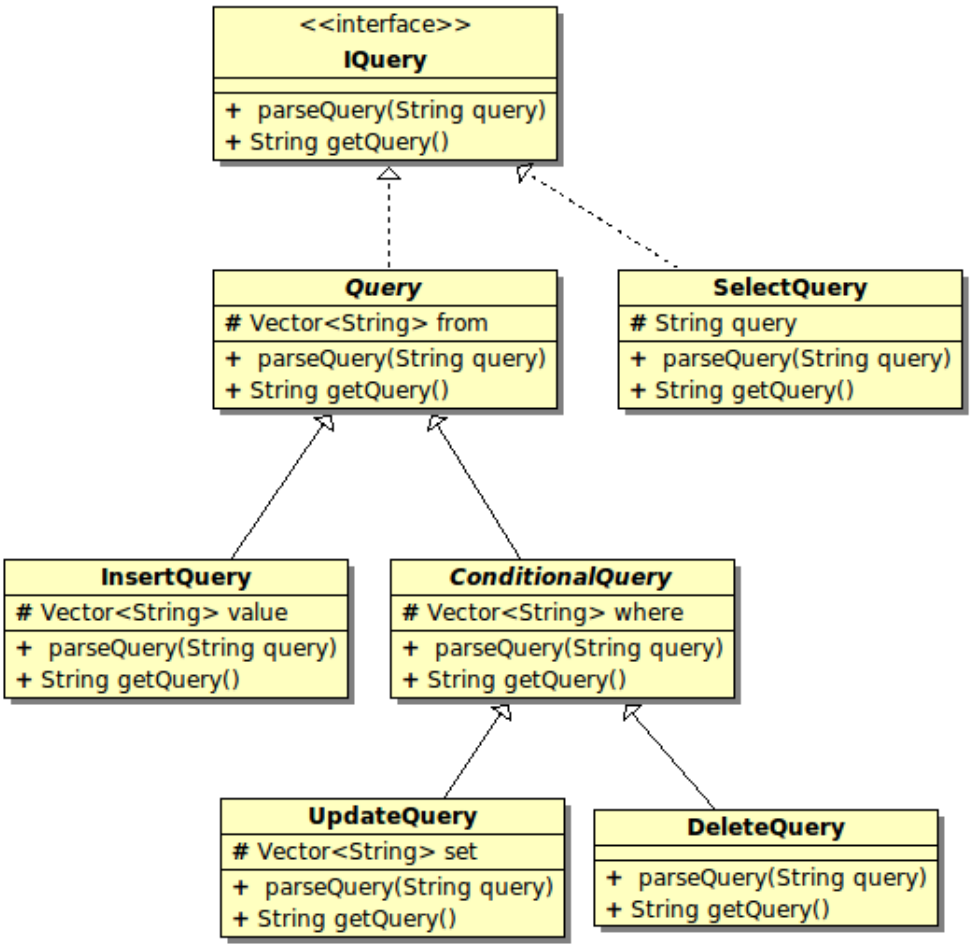

FIGURE 2 – Diagramme de classes : Query

2.3 Bases de données

Pour la distribution de nos données, nous avons décidé de partager nos données verticalement :

- les facteurs de transmissions d'une maladie
- les virus à l'origine d'une maladie
- le nom des maladies
- le nombre de cas recensés d'une maladie par zone, cette base de donnée sera une base virtuelle et sera donc découpée horizontalement par zone OMS (ici nous en prendrons deux : Europe et Europe de l'est)

Ce qui donne le shéma de tables suivant :

2.4 Dérivation puis Intégration

Maintenant que nous avons réparties nos données sur différents SGBD avec une approche dite de dérivation. Nous devons nous demander comment allons nous rendre possible la re-

cherche sur ces différentes bases. Deux méthodes s'offrent à nous : nous continuons à dériver, ou nous intégrons nos différents schémas de bases.

La première solution revient à découper notre schéma global de façon à ce qu'il colle à nos schémas locaux : donc à chaque base de données. Cette solution est la plus simple des deux mais nécessite de gros efforts d'implémentation puisque le schéma global doit être découpé en schémas locaux "à la main" pour chaque base de données. De plus cette solution devient inmaintenable lors du changement de schémas de base : en d'autres termes il faut se replonger dans la programmation.

La deuxième revient à prendre tous les schémas locaux pour faire émerger un schéma global. Cette solution est bien plus compliquée au niveau de la réflexion et de la logique qu'il y a derrière. Mais la généricité de l'implémentation permet de perdre moins de temps en cas de changement des schémas des bases de données. En effet nous avons imaginé une solution entièrement basée sur les méta-tables, ce qui nous permet d'abstraire tout notre code afin de le rendre réutilisable et modulable. Nous avons donc en toute logique retenu cette deuxième solution.

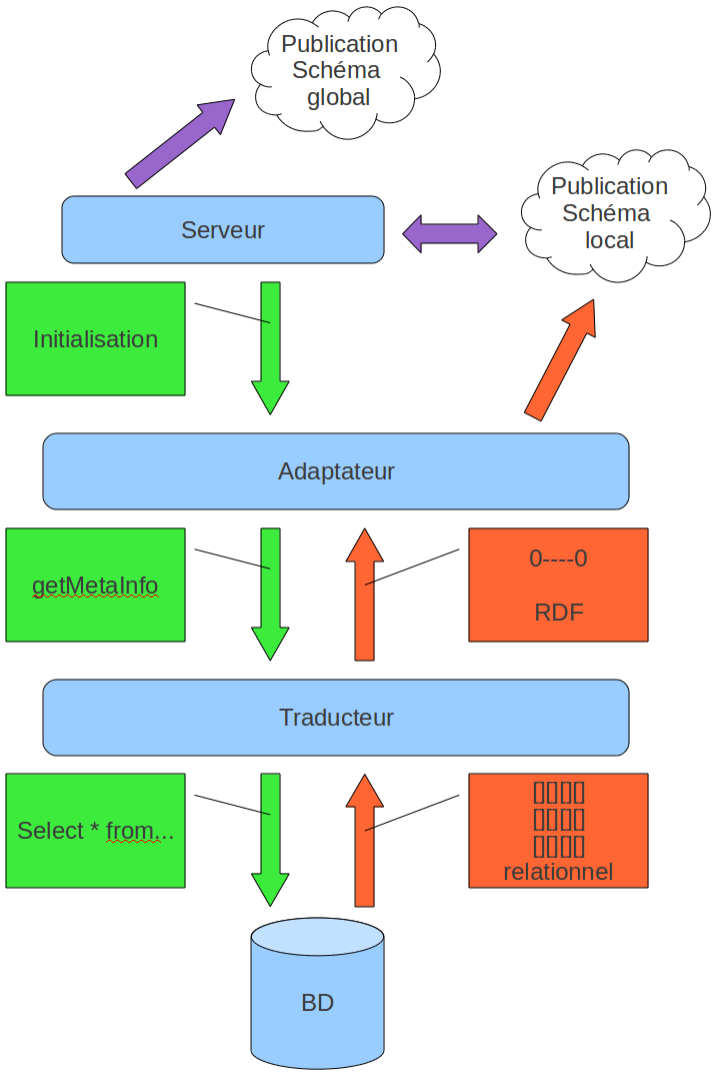

FIGURE 3 – Schéma d'initialisation serveur

2.5 Intégration

Puisque notre système repose sur les méta-informations des schémas locaux, le problème suivant est de pouvoir les rassembler pour faire émerger un schéma global correspondant à toutes les informations accessibles par un utilisateur.
Pour réaliser cela nous avons immédiatement choisi la technologie RDF. En effet elle nous permet de représenter nos données sous forme de triplets : deux entités et une relation entre eux. Ce système nous permet d'ajouter à volonté des relations de tous types entre toutes nos entités. Vous comprendrez donc qu'il devient facile de regrouper des entités de différentes bases de données pour faire émerger notre fameux schéma global, en précisant que les données d'une base sont équivalentes au même données d'une autre.
De plus RDF nous est fourni avec plusieurs outils très utiles :

- D2RQ : traducteur d'objet relationnel en objet rdf.
- N3 : cette notation qui va de paire avec D2RQ permet de lier des bases de données avec des objets rdf. En décrivant le mappage entre les tables et colonnes d'un côté et les types RDF de l'autre, D2RQ peut construire tout le schéma RDF correspondant à une base de données. Nous utiliserons donc D2RQ comme traducteur pour toutes nos bases de données.

- SparQL : ce langage permet de réaliser très simplement des requêtes complexes dans nos triplets RDF. Et notamment de faire émerger des connaissances plus subtilement que par des recherches "classiques" basées sur la comparaison de valeurs.

Le seul problème de D2RQ est qu'il permet de faire tout ce travail qu'en sélection. En effet nous ne pouvons pas insérer, mettre à jour ou supprimer des données d'une base. Il faudra donc trouver un autre moyen.

Nous nous servirons donc de RDF aussi bien pour réaliser les requêtes de sélection de nos utilisateurs, que pour diriger nos requêtes vers telle ou telle base de données en fonction des schémas locaux. En effet puisque SparQl nous permet de récupérer sous forme RDF les informations des tables, nous pourrons construire un schéma ontologique regroupant les méta-informations des bases de données. Celui-ci pourra nous renseigner sur la localisation des informations.

2.5.1 Schéma ontologique des Maladies de leur transmission et de leur facteur

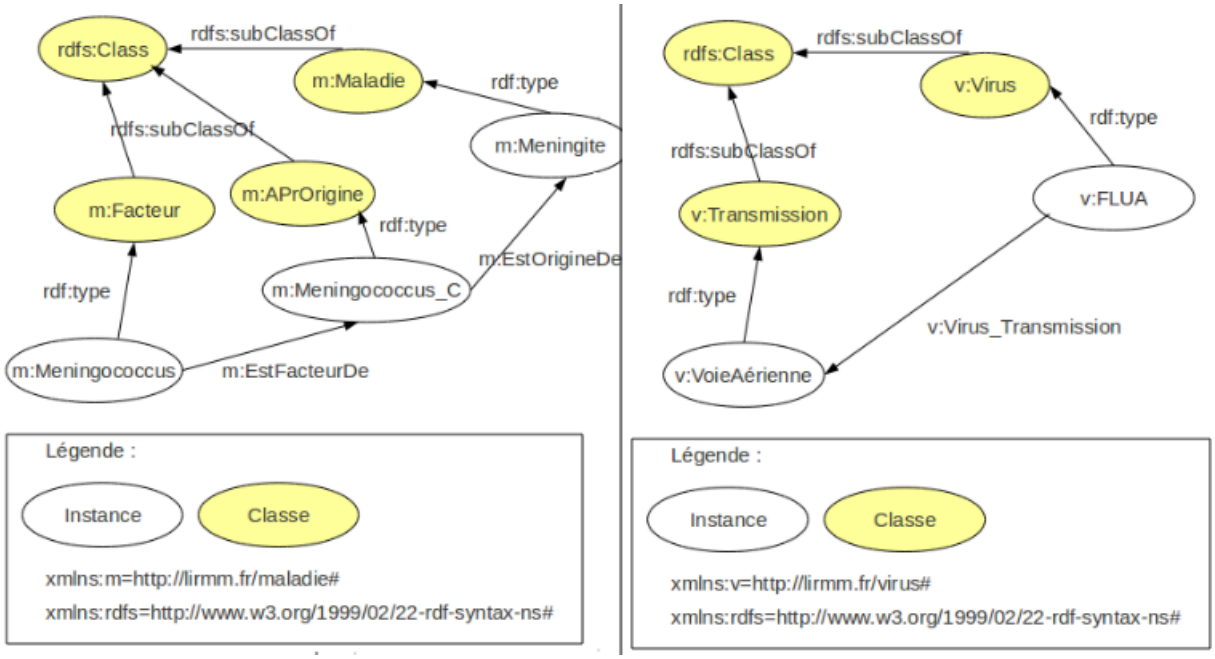

FIGURE 4 – Schéma RDF de la base MaladieFacteur et de la base VirusTransmission

Nous avons ainsi deux bases de données pour lesquelles : un facteur de type "virus" est à l'origine d'une maladie virale et un facteur de type "bactérie" est à l'origine d'une maladie bactérienne. Il semble alors normal de vouloir associer la table virus à la table facteur, lorsque le type de facteur est "virus". Ainsi, pour faciliter la création du modèle ontologique, on rajoute dans le mapping, une classe : FACTEURVIRAUX (respectivement FACTEURBACTERIENS), puis il suffit de dire que la classe virus est équivalente à la classe FACTEURVIRAUX dans notre modèle ontologique. On obtient ainsi la logique descriptive suivante :

VIRUS $\subseteq FACTEURVIRAUX$
$BACTERIE \subseteq FACTEURBACTERIENS$
$FACTEURVIRAUX \subseteq FACTEUR$
$FACTEURBACTERIENS \subseteq FACTEUR$
$MALADIEVIRALE \subseteq MALADIE \cap (\geq APRORIGINE)(\exists APRORIGINE.VIRUS)$
$MALADIEVIRALE \subseteq MALADIE \cap (\geq APRORIGINE)(\exists APRORIGINE.BACTERIE)$

Nous obtenons ainsi le graphe RDF correspondant et grâce auquel nous allons construire notre modèle ontologique reliant les facteurs de la base MaladieFacteur et les virus de la base VirusTransmission :

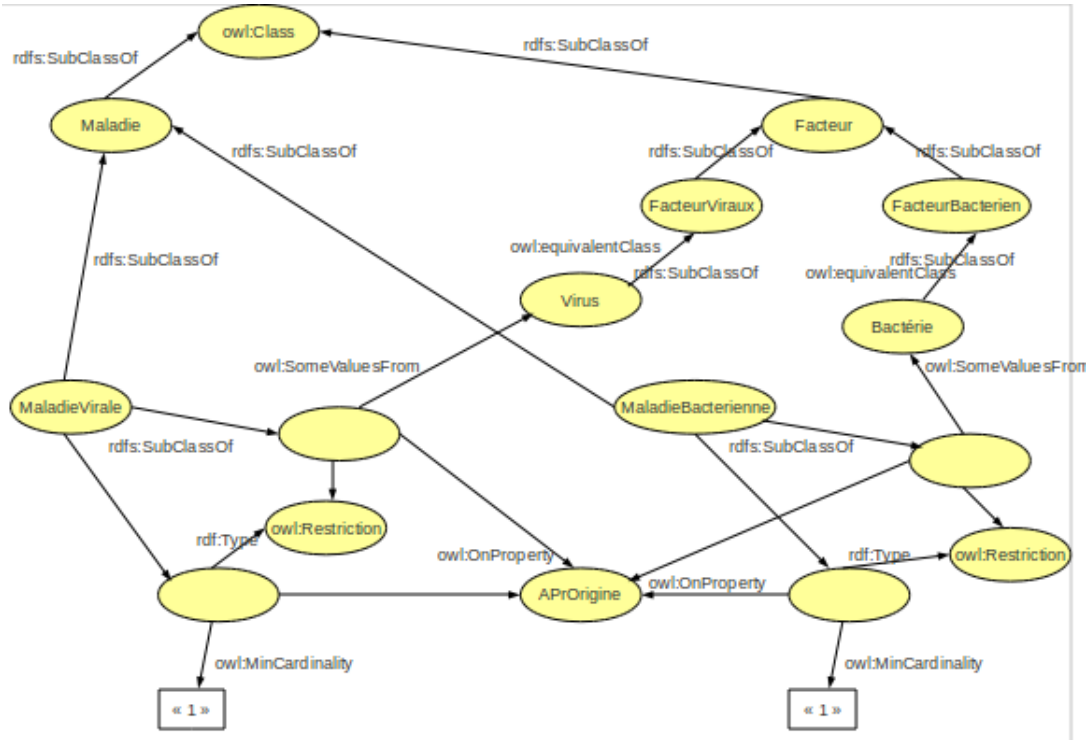

FIGURE 5 – Schéma ontologique Maladie

2.5.2 Schéma RDF Recensement

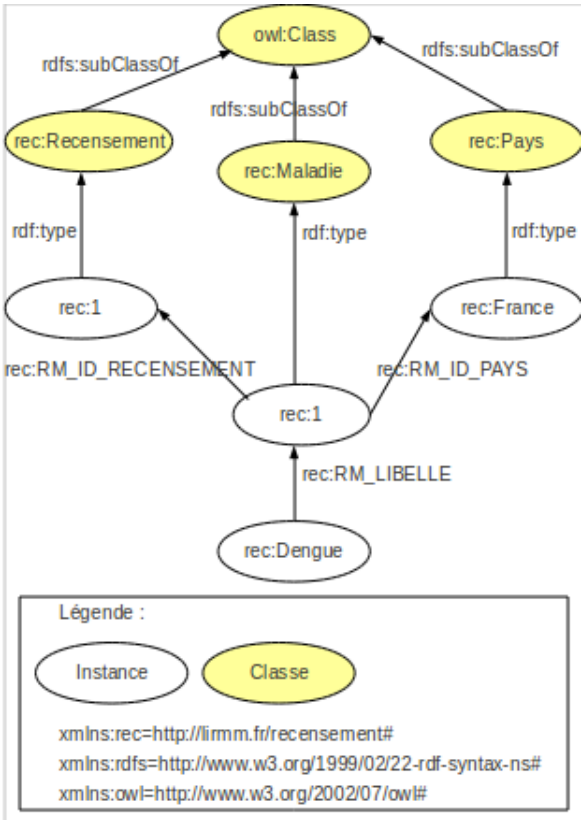

FIGURE 6 – Schéma ontologique de la base recensement

2.5.3 Schéma RDF Global

Comme nous pouvons le voir sur le schéma ci-dessus, la base de recensement possède également une table Maladie. Il est alors logique de dire que cette table et la table Maladie de la base MaladieFacteur sont équivalentes, pour cela, nous utilisons la propriété equivalentClass du langage owl, dans la construction de notre modèle ontologique. Nous obtenons alors le modèle ontologique suivant :

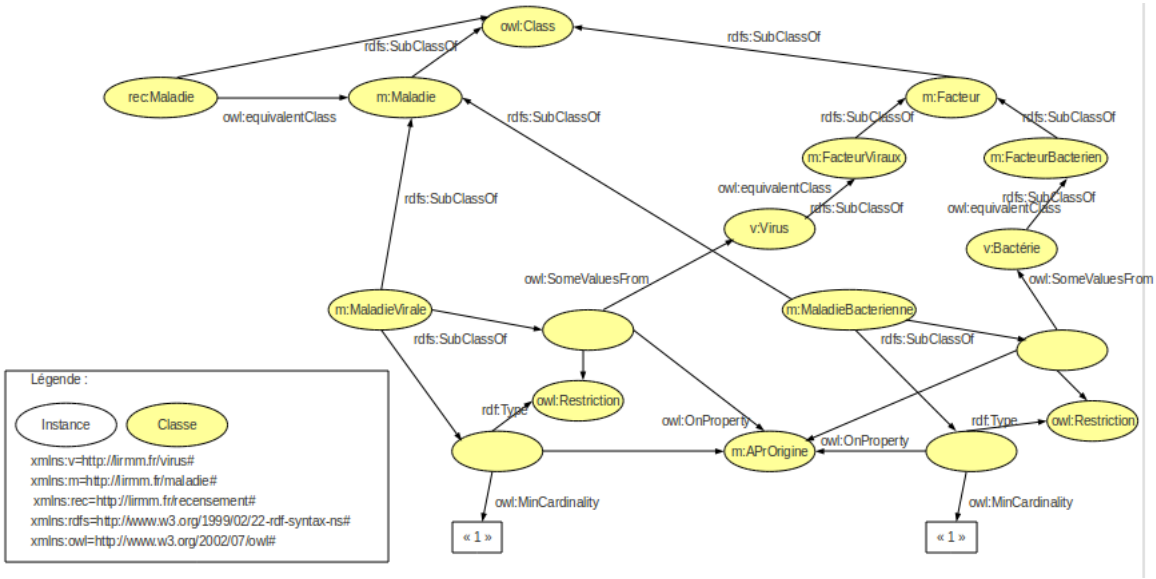

FIGURE 7 – Schéma global du modèle ontologique

2.6 Insertion, Modification et Suppression

Le dernier problème restant est donc de savoir comment nous allons insérer, modifier ou supprimer nos données des bases. Pour cela nous utiliserons la technique classique des traducteurs. Il y aura donc un traducteur par SGBD différent afin de faire correspondre la requête entrante aux normes de la base de données.

3 Modèle de conception

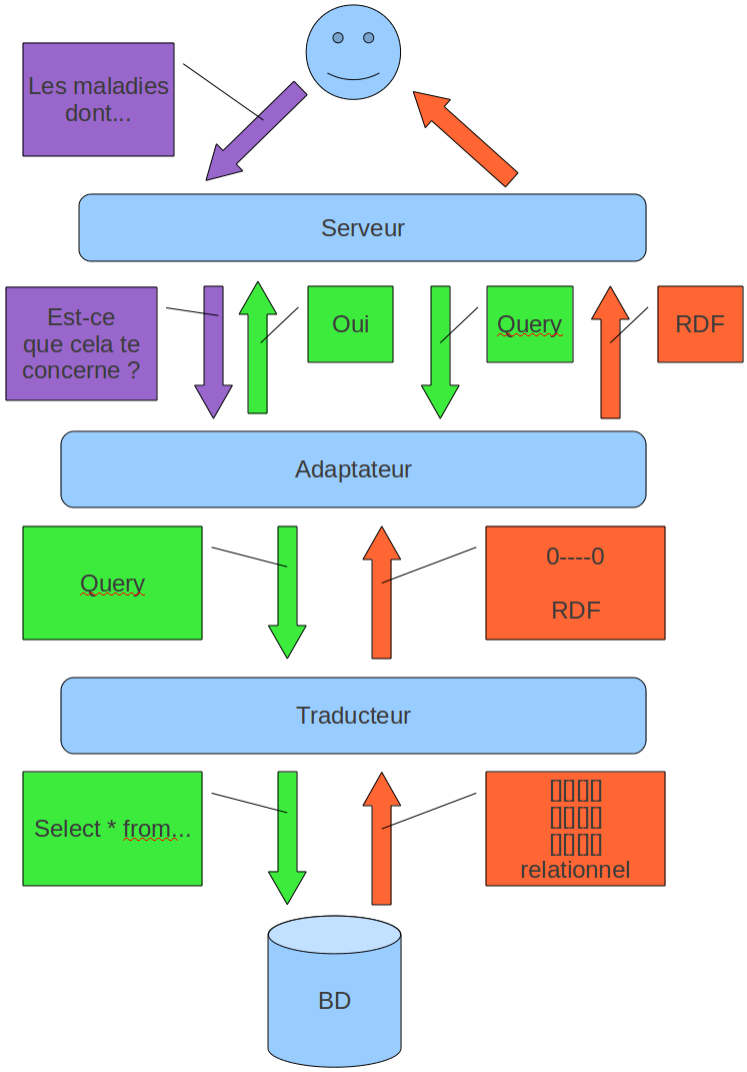

FIGURE 8 – Schéma de requête

3.1 Serveur pseudos-médiateurs

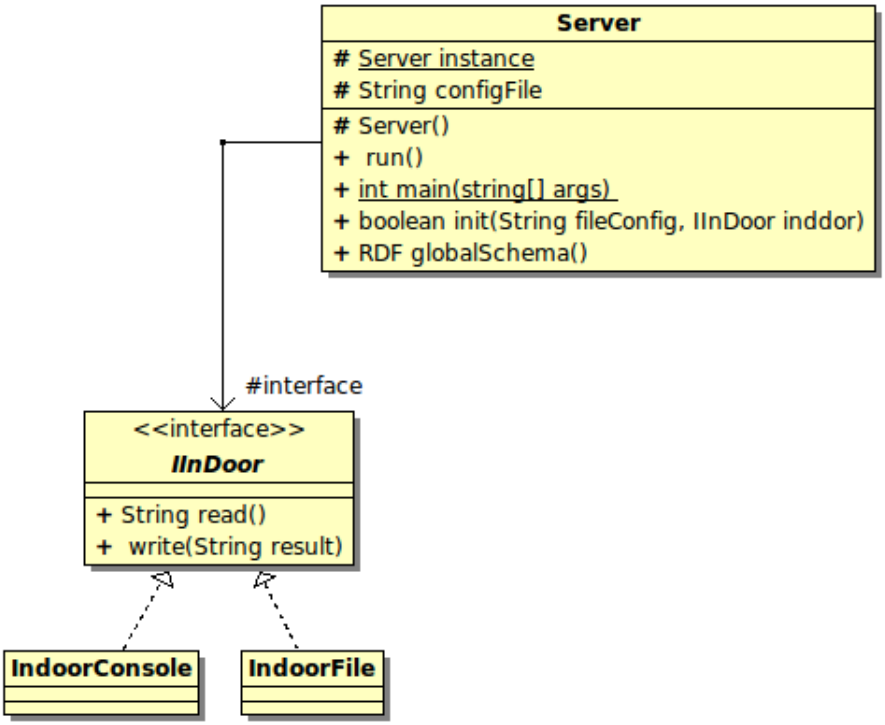

FIGURE 9 – Diagramme de classes : Server

Comme nous pouvons le voir sur ce schéma, le serveur sera le premier maillon de la chaine à recevoir la requête de l'utilisateur. Il aura alors la charge de transformer la requête qui arrive sous forme de chaine de caractères en type Query. Pour cela nous mettrons en place une fabrique à Query. Ensuite le serveur demandera à l'adapteur principal si la requête le concerne, si c'est le cas il lui transmettra la partie de la requête qui lui correspond.

Il n'aura plus alors qu'à attendre le résultat, qui lui sera retourné par l'adaptateur, et au besoin transformer celui-ci du format rdf au format souhaité.

Puisque notre serveur est au centre de la recherche : il prend la recherche et retourne le résultat, nous avons décidé de le caractériser par le nom pseudo-médiateur. En effet dans une architecture classique le médiateur est celui qui fractionne les requêtes pour que celles-ci correspondent au schéma local des adaptateurs. C'est ce que fait notre serveur, mais un médiateur au sens propre du terme, à d'autres exsigences que nous ne traiterons pas dans ce projet, d'où le nom de pseudo-médiateur.

3.2 Adaptateurs

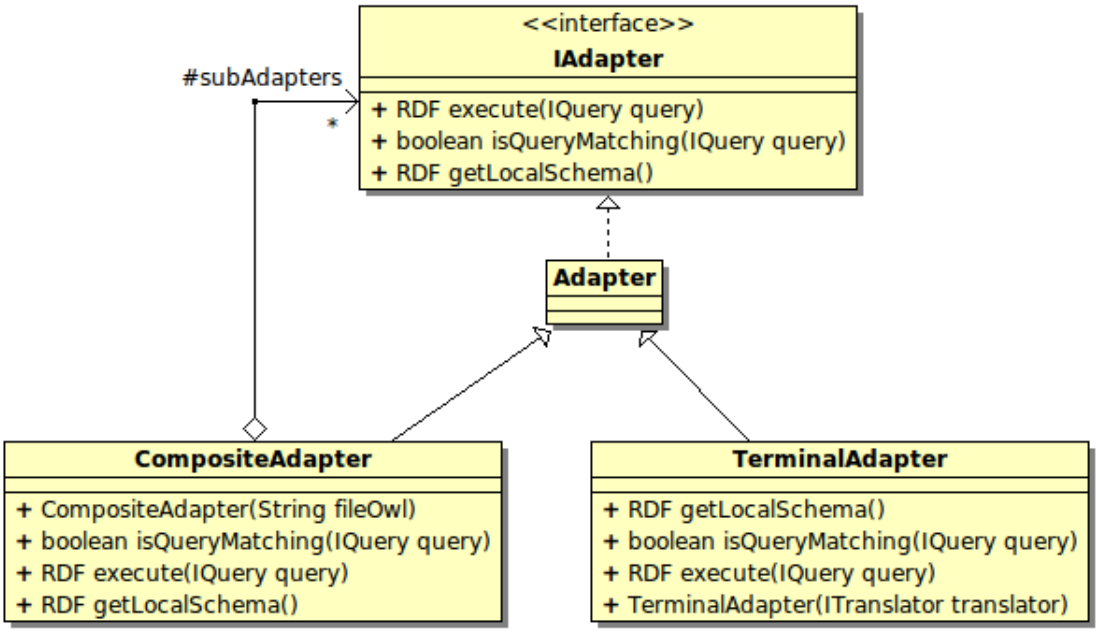

FIGURE 10 – Diagramme de classes : Adapter

Nos adaptateurs sont les pièces principales du projet. Ils devront publier le schéma local des bases de données dont ils ont la responsabilité. Ces schémas seront sous la forme RDF pour faciliter la recherche. Ils contiendront le noms des tables et des colonnes des bases.

Pour faciliter l'implémentation nous avons immaginé deux types d'adaptateur : les composites et les finaux. Les adaptateurs composites, sont des adaptateurs qui ont à leur charge non pas une base de données mais des adaptateurs qui peuvent être eux aussi composites ou finaux. Ces adaptateurs devront donc pouvoir faire émerger un schéma rassemblant l'ensemble de leur sous-adaptateurs.

Puisqu'un adaptateur permet de faire tout cela, nous avons décidé d'utiliser un adaptateur composite dans notre serveur. Celui-ci devra alors faire émerger le schéma global de nos bases de données.

3.3 Traducteurs

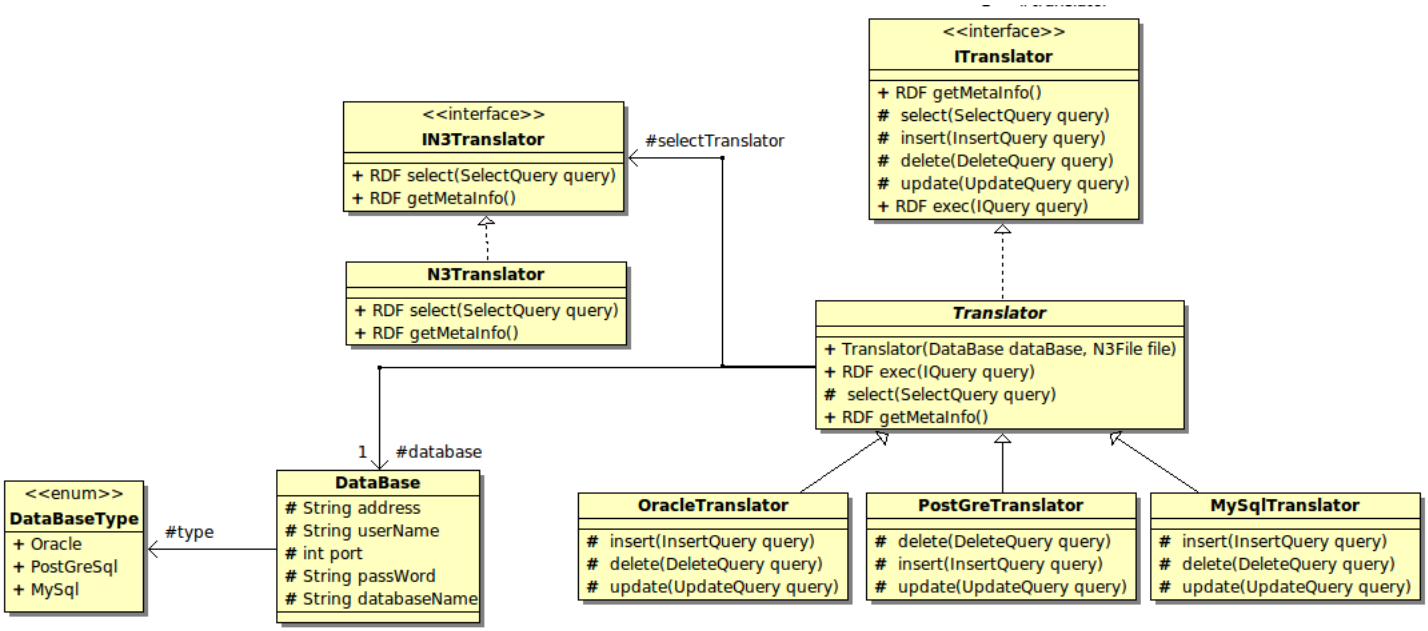

FIGURE 11 – Diagramme de classes : Translator

Comme son nom l'indique un traducteur permet tout simplement de convertir une requête de notre système en requête compréhensible pour le SGBD. Dans le cas de la sélection nous utiliserons D2RQ comme traducteur. Pour tout le reste nous l'implémenterons nous même.

4 Modèle d'implémentation

4.1 Initialisation

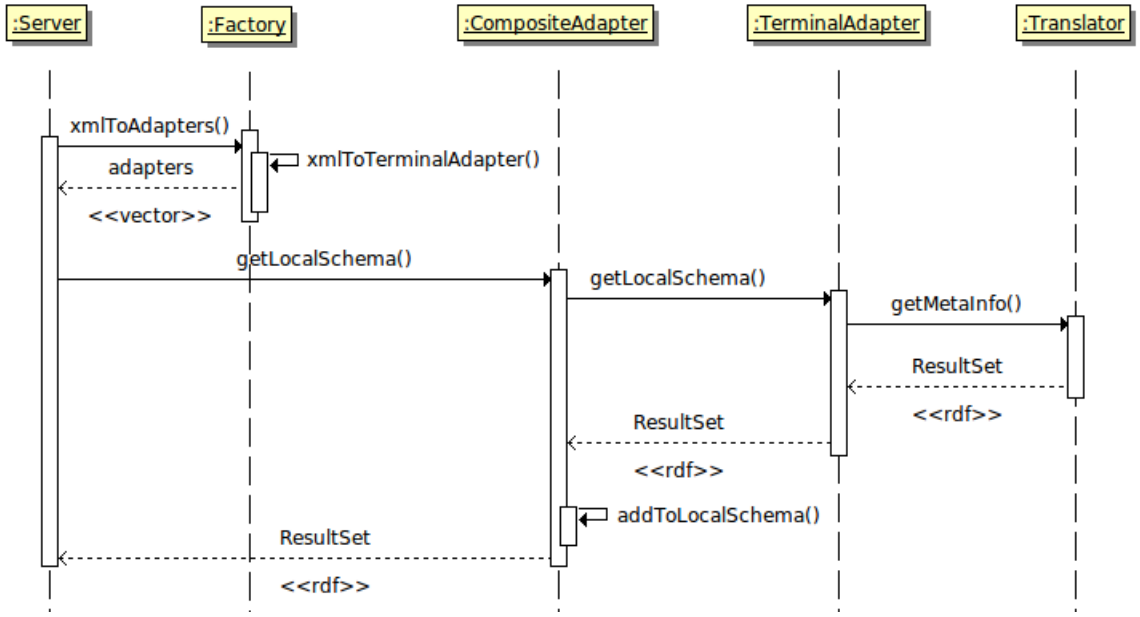

FIGURE 12 – Diagramme de séquence : Initialisation

Lorsque le serveur est instancié il charge le fichier de configuration en xml et créé un adaptateur composite. Celui-ci va chercher dans le fichier de configuration les informations nécessaires à la mise en place des sous-adaptateurs. Chaque adaptateur composite devra avoir un fichier owl pour lui signifier l'ontologie permettant de créer le schéma global. Ce fichier ne devra comporter que les points de liaisons des divers schémas locaux. Il construira alors les sous-adaptateurs.

Lorsque l'adapteur est terminal un fichier N3 lui sera alors fourni pour qu'il puisse construire son traducteur. De plus il devra avoir la configuration nécessaire à la création d'une DataBase. Il pourra donc utiliser la fabrique pour créer le bon traducteur.

4.2 Requête

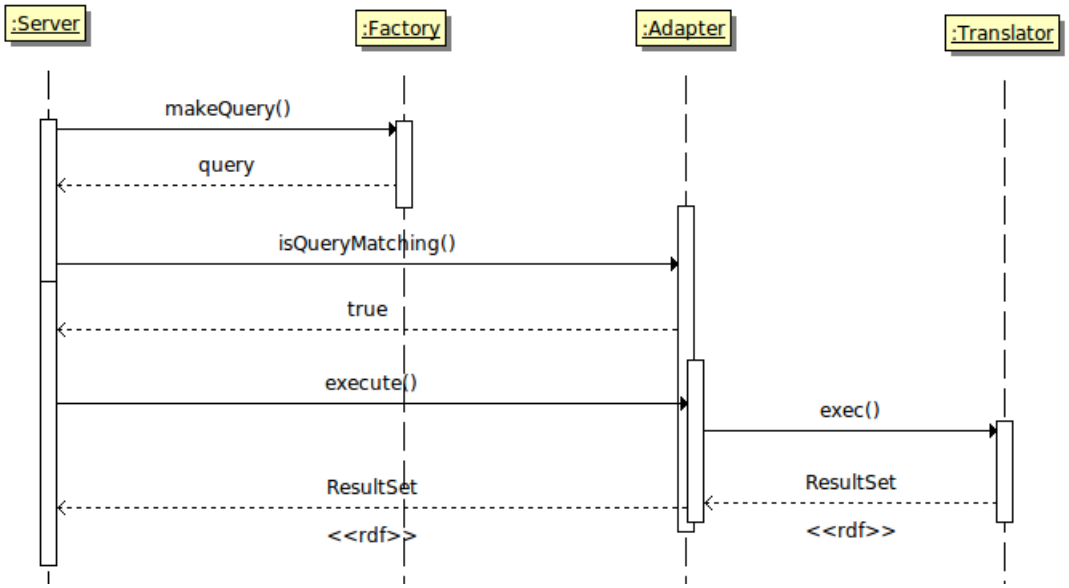

FIGURE 13 – Diagramme de séquence : Initialisation

Lorsque tout est initalisé le serveur rentre dans sa boucle run(), cette méthode doit appeler successivement interface.read() et afficher le résultat en appellant interface.write().

Lorsqu'une requête arrive le serveur doit construire une IQuery à l'aide de la fabrique. Il n'a ensuite qu'à appeler mediatorLike.exec(). Celui-ci va appeler sur tous ces sous-adaptateurs isQueryMatching() et si c'est vrai il appellera alors exec() de celui-ci.

Lorsqu'un traducteur final sera appelé il passera la requête à son traducteur par le biais de la fonction exec(). Celui-ci devra tout d'abord vérifier de quel type est la requête, si elle de type select le traducteur N3 sera appelé sinon ce sera la fonction abstraite, qui aboutira dans le bon traducteur correspondant au bon sgbd.

Lorsque le résultat de la requête sera retourné à l'adaptateur-composite celui-ci devra ajouter à son modèle ontologique les triplets récupérés (toujours dans le cas où c'est un select). En remontant toute la hiérarchie notre modèle ontologique global se construira. Et ainsi nous retournerons le résultat de la requête à l'utilisateur.

4.3 Fabrique

Nous avons mis en place une fabrique pour faciliter la création des divers objets : base de données, requêtes, traducteurs et adaptateurs. En effet chaque instance de ces objets seront presques équivalentes. Les différences seront chargées par le biais d'un fichier xml ce qui permettra de paramétrer notre système.

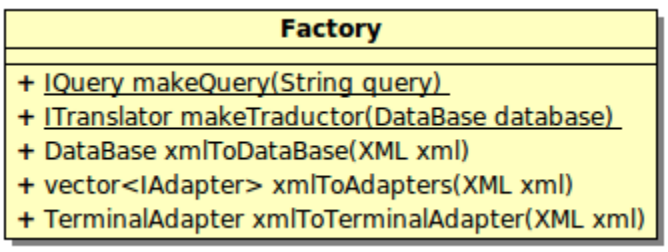

FIGURE 14 – Diagramme de classe de la fabrique

5 Glossaire

- **SGBD** : Système de gestion de bases de données.
- **schéma global** : le schéma de nos tables et de nos colonnes comme s'il ne s'agissait que d'une seule base de données.
- **schéma local** : le schéma physiquement stoqué sur une base de donnée particulière.
- **méta-informations** : les informations concernant le nom des tables et des colonnes d'une base de donnée.
- **requête** : dans notre système nous parlons de deux sortes de requêtes différentes, les requêtes du client donc qui arrivent dans notre serveur sous forme de chaine de caractères et les requêtes internes qui sont ces même requêtes mais transformées en objet.

6 Annexes

6.1 Diagramme UML

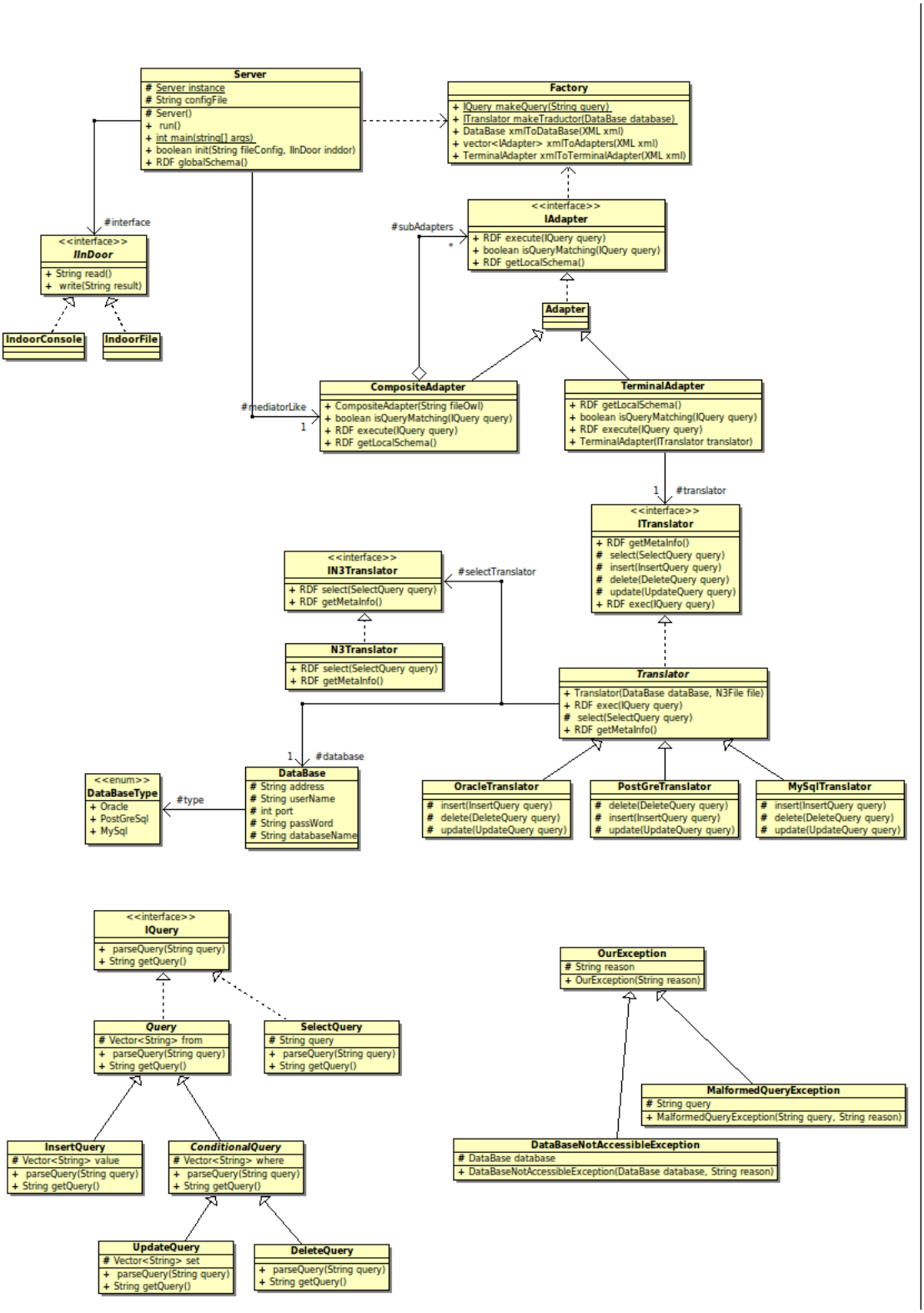

FIGURE 15 – Diagramme de classes

6.2 Schéma des tables des bases de données

Schéma des tables

ORACLE

TRANSMISSION(VT_MODET);
VIRUS(VV_ABREV, VV_NOMV, VV_TYPEGENOME, VV_NOMFA);
VIRUS_TRANSMISSION(#VVT_MODET, #VVT_ABREVV);

SQL

MALADIE(MM_NOM_MALADIE);
FACTEUR(MF_ABREV_FACTEUR, MF_NOM_FACTEUR, MF_TYPE_FACTEUR);
APRORIGINE(#MA_NOMM, #MA_ABREVF, MA_SOUCHE);

POSTGRE

RECENSEMENT(RR_ID, RR_NBRE, RR_ANNEE);
ZONE_OMS(RZ_LIBELLE , RZ_POPULATION);
PAYS(#RP_LIBELLE_ZO, RP_LIBELLE, RP_POPULATION);
MALADIE(RM_ID, RM_LIBELLE, RM_POPULATION, #RM_ID_RECENSEMENT, #RM_LIBELLE_PAYS);

FIGURE 16 – Schéma des tables

6.3 XML de configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<raja>
  <CompositeAdapter>
    <OWL url="" />
      <CompositeAdapter>
        <OWL url="" />
          <TerminalAdapter>
            <Traducteur>
              <N3>
                <URL></URL>
                <GetMetaInfo></GetMetaInfo>
                <Prefix>
                  <P></P><P></P><P></P>
                </Prefix>
              </N3>
              <Database>
                <dataBaseName></dataBaseName>
                <address></address>
                <userName></userName>
                <port></port>
                <password></password>
                <type></type>
              </Database>
            </Traducteur>
```

```
      </TerminalAdapter>
      <TerminalAdapter>
        <Traducteur>
          <N3>
            <URL></URL>
            <GetMetaInfo></GetMetaInfo>
            <Prefix>
              <P></P><P></P><P></P>
            </Prefix>
          </N3>
          <Database>
            <dataBaseName></dataBaseName>
            <address></address>
            <userName></userName>
            <port></port>
            <password></password>
            <type></type>
          </Database>
        </Traducteur>
      </TerminalAdapter>
  </CompositeAdapter>
  <CompositeAdapter>
    <OWL url="" />
    <TerminalAdapter>
      <Traducteur>
        <N3>
          <URL></URL>
          <GetMetaInfo></GetMetaInfo>
          <Prefix>
            <P></P><P></P><P></P>
          </Prefix>
        </N3>
        <Database>
          <dataBaseName></dataBaseName>
          <address></address>
          <userName></userName>
          <port></port>
          <password></password>
          <type></type>
        </Database>
      </Traducteur>
    </TerminalAdapter>
    <TerminalAdapter>
      <Traducteur>
        <N3>
          <URL></URL>
          <GetMetaInfo></GetMetaInfo>
          <Prefix>
            <P></P><P></P><P></P>
```

```
            </Prefix>
          </N3>
          <Database>
            <dataBaseName></dataBaseName>
            <address></address>
            <userName></userName>
            <port></port>
            <password></password>
            <type></type>
          </Database>
        </Traducteur>
      </TerminalAdapter>
      <TerminalAdapter>
        <Traducteur>
          <N3>
            <URL></URL>
            <GetMetaInfo></GetMetaInfo>
            <Prefix>
              <P></P><P></P><P></P>
            </Prefix>
          </N3>
          <Database>
            <dataBaseName></dataBaseName>
            <address></address>
            <userName></userName>
            <port></port>
            <password></password>
            <type></type>
          </Database>
        </Traducteur>
      </TerminalAdapter>
    </CompositeAdapter>
  </CompositeAdapter>
</raja>
```