Université de Namur
Faculté d'informatique
Année académique 2017–2018

Archipelago : Un framework de peristence pour bases de données orientées graph.

Gilles Bodart

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
CLEVE Anthony

Co-promoteur : LAMBIOTTE Renaud

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Contents

Introduction

De nos jours, les unités de stockages sont de plus en plus accessible au grand public, les différentes entreprises impliquées dans le développement des systèmes de gestion de base de données l'ont bien compris. Depuis plusieurs années, L'hégémonie des bases de données relationnelles se fait de moins en moins écrasante, le solutions alternatives du NoSql séduisent jours après jours de grandes entreprises.

La grande force des SGBDR provient des fondement théorique mis en place en 1970 par Edgar Frank Codd, IBM défini le langage "Structured Query Language" (SQL) pour utiliser ce système. Cette normalisation de la représentation de l'information génère un sentiment de sûreté pour les architectes de logiciels.

A contrario, le mouvement NoSql pour "Not Only SQL" ne se base pas sur un fondement théorique commun. Que ce soit la théorie des graphs, les systèmes de hash key/value avec partitions, ... chaque système choisit ce sur quoi il va se baser. Mais si il existait une sorte de contrat liant les SGBD NoSql entre eux afin d'assurer une cohérence d'utilisation, ce mémoire va se concentrer sur les implémentations de la théorie des graph, en essayant de mettre en place un système de comparaison, nous allons tenter de spécifier certains objectifs commun et développer un framework Java permettant d'utiliser de manière agnostique l'une ou l'autre base de donnée.

Part I

État de l'art

Chapter 1

L'évolution du NoSql

Un SGBD est par définition un ensemble de procédés permettant d'organiser et de stocker des informations (potentiellement de gros volumes). Si stocker et retrouver l'information est un des plus grand challenge d'un SGBD, une communauté de développeur, pensent que ces système devraient pouvoir offrir d'autres fonctionnalités.

A partir des années 1980, le modèle relationnelle supplante les autres formes de structures de donnée.

Les évolutions logicielles suivant naturellement les évolutions matérielles, la généralisation des interconnexion des réseaux, l'augmentation de la bande passante, la diminution du cout des machines, la miniaturisation des espaces de stockage, ... de nouvelles opportunités sont arrivé au XXI^{e} siècle.

Les entreprises comme Google, Amazon, Facebook, Twitter, ... sont tour à tour arrivés aux limites du modèle Relationnel. Que ce soit a cause de volumes astronomiques (plus de 100 pétaoctets) ou du nombre de requêtes par secondes, il fallut développer une nouvelle façon de gérer les données.

Le NoSql découle de ce genre de problèmes, ces modèles arrivent avec des approche optimisée pour des secteurs spécifiques.
Comme ces modèles représentent ce qui n'est pas Relationnel, par soucis de classification, nous allons distinguer 4 usages principaux :

- Performances : L'objectif du SGBD sera d'augmenter au maximum les performances de la manipulation des données.
- Structures simples : Pour s'affranchir de la rigidité du modèle relationnel, la structure sera généralement simplifiée, en utilisant une représentation plus souple comme le JSON par exemple.
- Structures spécifiques : Certain moteur NoSql sont liés a des besoins spécifiques, la structure de représentation de donnée sera dès lors focalisée sur un cas d'utilisation.
- Volumétries : Un des principal aspect important des SGBD NoSql est leur capacité de gérer la montée en charge de données. La distribution des traitements au travers de plusieurs clusters est un facteur très important dans la plupart des applications BigData.

Et nous allons aussi distinguer 4 grandes familles de représentation de Schéma de données :

- Document : L'utilisation de format spécifiques tels que le très rependu JSON permet de

stocker les données sur base de fichier.
- Clé / Valeur : Le système le plus simple, il manipule des paires de clé/valeurs, ou accède à un élément en fonction d'une table de hachage.
- Colonne : Inspiré de Google BigTable, la structure ressemble à la table relationnelle. On peut la comparer à une table de hachage qui va référencer une ou plusieurs colonnes.
- Graph : La famille Graph se distingue du fait que les entités ne sont pas considéré comme des entités indépendante, mais que la relation être ces objets est tout aussi important que le contenu.

Les implémentations de bases de données de types graph sont de plus en plus nombreuses, les relations entre les éléments permettent de parcourir le graph de manière très performante les rendent de plus en plus intéressante pour les entreprises possédant des millions de données. L'utilisation de ce genre de SGBD est dès lors tout a fait recommandé pour des entreprises intéressé entre les relations de ces donnés tels que des profils sociaux, des liens de cause a effet, des liens géographiques et bien d'autre.

Pourquoi utiliser une BDOG plutôt qu'une BD relationnelle comme Oracle ou MySql ?

L'utilisation des SGBD relationnelle pour tout type de problème est révolue. Grâce à l'apparition de SGBD NoSQL spécifique permet dès à présent une approche plus personnalisée. En effet, comme tout système, le relationnelle a ses limites, l'approche actuelle des entreprises est plus orienté "Big Data", elles veulent tout stocker afin d'avoir le plus d'informations possible, or du fait que le relationnelle cadenasse les données dans une table préalablement définie, on se doit de tronquer l'information ou alors, mettre a jour le schéma de définition des tables. Dans une BDOG tel que Neo4J, nous pouvons ajouter toutes les informations que nous voulons sans condition préalable, cette approche oriente plus un contrôle de l'intégrité des donnée aux applications.

Si l'objectif de l'applicatif est de représenter un système récursif comme une arborescence de fichier, de la généalogie, ... le modèle relationnelle se basera sur une table faisant référence à elle même, l'utilisateur devra donc réaliser une jointure par profondeur, si l'arborescence descend jusqu'à 15 niveau, cela peut devenir problématique. L'approche BDOG permet de parcourir une arborescence sans pour autant charger l'ensemble des nœuds ayant le même label, plus besoins de projection ensembliste pour pouvoir continuer le chemin.

En relationnel, il existe de nombreux moyen d'historiser les données, cependant, lorsque le schéma comprend des relations cycliques, cette opération devient plus complexe. L'utilisation d'un BDOG rend ça plus simple, il suffit de créer un nouveau noeud avec les anciennes données, lier ce noeud avec une relation historique en y spécifiant la date de mise à jour et ensuite changer les valeur du noeud référencé par le cycle. Ce procédé peut être mis au point dans n'importe quelle représentations et ne nécessite aucune refonte globale du modèle de donnée.

On parle de Base de donnée orientée graph mais quelle est la différence entre une relation entre deux noeud et une relation entre deux table ?

Les relations entre deux tables se font à l'aide de clé étrangère définie lors de la création du schéma de données elles apporte de l'information supplémentaire en permettant de réaliser des jointure entre plusieurs tables.

La relation dans les BDOG est une information à part entière, elle peut posséder autant de donnée que le noeud lui même, la force de ce genre de système est de pouvoir lier deux noeud distinct par n'importe quelle relation et ce à tout moment.

Chapter 2

Backgroud technique

2.1 Neo4J

2.1.1 Description

Créé par Neo Technologie, une sociétée suédo-américaine, elle est actuellement (selon db-engines.com) la base de donnée orienté graph la plus utilisée dans le monde. Développé en java sous licence GPL V3, AGPL ou licence commerciale, Neo4J représente les données sous formes de "Noeuds" et de "Relations", chacun de ces éléments peuvent contenir une ou plusieurs propriétés. Les propriétés sont des couples clés/valeurs de type simple, comme des chaines de caractères ou des valeurs numériques, des coordonnées spatiales, ...

Une des particularité de Neo4J est l'absence de structure définie, un noeud peut être labellisé afin de permettre de travailler sur un ensemble d'éléments, mais il n'y aura aucune contrainte sur les propriétés du noeud. Cette particularité rend ce SGBD bien adapté pour les modèles évoluant fréquemment.

2.1.2 Langage de requête

Le langage propre à Neo4J se nomme "Cypher", il a pour but de réaliser plus simplement que SQL les opérations de parcours ou d'analyse de proximité.

```
1 CREATE (mamours:Person {name:"Mamours"})
2 CREATE (mamyco:Person {name:"Mamyco"})
3 CREATE (gilles:Person {name:"Gilles"})
4 CREATE (marie:Person {name:"Marie"})
5 CREATE (bebe:Person {name:"Enfant"})
6 CREATE (mamours)-[:PARENT_OF]->(gilles)
7 CREATE (mamyco)-[:PARENT_OF]->(marie)
8 CREATE (marie)-[:PARENT_OF]->(bebe)
9 CREATE (gilles)-[:PARENT_OF]->(bebe)
```

Ces requêtes vont créer 5 noeud et 4 relations PARENT_OF, nous pouvons aisément comprendre que "Mamyco" est parente de "Marie"

```
MATCH (p:Person)-[:PARENT_OF]->(c:Person)
RETURN DISTINCT (p)
```

Cette query va retourner tout les nœuds distinct qui ont une relation :PARENT_OF avec un autre noeud.

```
MATCH (gp:Person)-[:PARENT_OF*2]->(c:Person)
RETURN DISTINCT (gp)
```

Celle-ci quant à elle va retourner toutes les personnes qui sont parent de parent et donc grand parents.

ces deux exemples peuvent montrer la force de l'utilisation d'un SGBD de type graph pour représenter un ensemble hierarchique de données par rapport au SGBD relationnelles qui nécessiterai une double jointure sur la Table "Person".

2.1.3 Communication

Neo4J peut être utilisé sous plusieurs formes.

La première option est une solution embarquée, ce choix peut être très intéressant en alternatif au très célèbre SQLite relationnel.

La deuxième, pour toute application distribuée, est une solution autonome pouvant tourner comme un service sur tout type de plateforme. Le protocol "Bolt", développé par "Néo Technologie" est grandement conseillé pour communiquer avec ces serveurs distants.

Son utilisation est simple grâce à l'utilisation de la libraire native à Néo4J pour java (neo4j-java-drive) ou avec l'utilisation de l'API Rest déployée en même temps que le SGBD.

2.2 OrientDB

2.2.1 Description

OrientDB est un SGBD initialement développé en C++(Orient ODBMS) ensuite repris en 2010 en Java par Luca Garulli dans une version multi-modèle sous licence Apache 2.0, GPL et AGPL. actuellement 3ème mondial (selon db-engines.com) il offre de nombreuses fonctionnalités intéressante.

OrientDB est base de donnée associant Document et Graph. Elle combine la rapidité et la flexibilité du type document ainsi que les fonctionnalités de relations des bases de données graph.

Ce SGBD est composé de trois grands éléments

- Document & Vertex : Source de contenu, ces élements peuvent être considéré comme des container de données, on peut le comparer avec la ligne d'une base de données relationnelle.

- Links & Edge : Une arrête orientés reliant deux éléments non nécessairement distinct.
- Property : Typée ou embarquée dans un document JSON, ceci va représenter le contenu de l'information. Ces propriétés sont bien entendu primordiales pour ordonner, rechercher, ...

Chaque Document ou Vertex appartient à une "Class", celle-ci peut être strictement définie ou plus laxiste. Comme dans la programmation orientée objet, OrientDB offre le principe de polymorphisme avec un système d'héritage entre les classes.

OrientDB vient dans sa version community avec un système de clustering permettant à l'utilisateur de correctement gérer les montées en charges. chaque document est identifié avec une partie désignant le cluster dans lequel l'information est stockée et une autre partie désignant sa position dans ce dernier (exemple @rid: 10:12). Chaque classe peut être associée a un ou plusieur cluster, permettant d'optimiser les accès dans des ensembles plus petits.

2.2.2 Langage de requête

OrientDB utilise une sorte de SQL avancée pour interpréter les requêtes. On peut de plus utiliser le langage Gremlin.

Voici quelques exemples d'utilisation du SQL avancé dans OrientDB.

```
CREATE CLASS Person EXTENDS V
CREATE CLASS Company EXTENDS V
CREATE CLASS WorkAt EXTENDS E
CREATE PROPERTY Person.firstname string
CREATE PROPERTY Person.lastname string
CREATE PROPERTY Company.name string
INSERT INTO Person(firstname, lastname) VALUES
    ("Gilles","Bodart"),("Marie","Van Cutsem")
       ou
INSERT INTO Company set name = "ACME"
```

Cet ensemble de requête ressemblant au langage SQL permet de créer deux vertex, Person et Company , un Edge WorkAt et leur associe certaine propriétés. Les deux types d'insert différent permettent comme en SQL d'ajouter un noeud.

```
SELECT FROM V
```

Metadata			Properties		
@rid	@version	@class	firstname	lastname	name
10:0	1	Person	Marie	Van Cutsem	
10:1	1	Person	Gilles	Bodart	
11:0	1	Company			ACME

Nous pouvons observer qu'OrientDB se charge de qualifier les document de plusieurs métadata en leur octroyant par exemple un identifiant unique (n° cluster:position), un numéro de version pour permettre une gestion de transaction "full optimistic"[1] et le class représente la structure du document.

[1] On laisse l'utilisateur continuer sa transaction et le refus se fera lors du commit si il y a eu une modification concurrente du même document

```
1 CREATE Edge WorkAt from 10:1 to 11:0
```

Cette petite requête va lier le document ayant l'id 10:1 au document dont l'id est 11:0 par une relation WorkAt. Nous pouvons traduire en français que Marie Van Cutsem travaille chez ACME.

2.2.3 Communication

OrientDB embarque une API Rest complète, toute actions pouvant être faite sur les interfaces web et consoles peuvent être reproduites au travers de cette API.

Chapter 3

Solutions existantes

3.1 Les librairies

3.1.1 Hibernate

Hibernate, soutenu et développé par JBoss, à mis au point un système de relation entre le code Java et le SGBD Néo4J, en réutilisant les mêmes annotations de relations que celles employées pour les bases de données relationnelles, à savoir :

- @OneToOne : Relation 1-1
- @ManyToOne : Relation n-1
- @OneToMany : Relation 1-n
- @ManyToMany : Relation n-n

Bien qu'à l'heure actuelle, aucune intégration avec OrientDB, un ticket est ouvert (OGM-855) depuis 2 ans auprès d'Hibernate.

3.1.2 Spring Data

Spring data à nouveau possède un module de communication avec le SGBD Néo4J, il intègre au célèbre Framework Spring les fonctionnalités de Néo4J-OGM[1], permettant de qualifier une classe avec certaines annotations facilitant la séréalisation et le déséréalisation vers la représentation graphique.

Exemple :

```
1 @NodeEntity(label="Film")
2 public class Movie {
3
4    @GraphId Long id;
5
6    @Property(name="title")
7    private String name;
8 }
```

[1] Object Graph Mapper

```
9
10 @NodeEntity
11 public class Actor extends DomainObject {
12
13    @GraphId
14    private Long id;
15
16    @Property(name="name")
17    private String fullName;
18
19    @Relationship(type="ACTED_IN", direction=Relationship.OUTGOING)
20    private List<Role> filmography;
21
22 }
23
24 @RelationshipEntity(type="ACTED_IN")
25 public class Role {
26     @GraphId
27     private Long relationshipId;
28     @Property
29     private String title;
30     @StartNode
31     private Actor actor;
32     @EndNode
33     private Movie movie;
34 }
```

3.1.3 Librairie Neo4J

Une librairie java développée par NeoTechnologie existe, elle permet de travailler aisément avec une base de donnée Neo4J embarquée ou à distance. Elle est en constante évolution et se trouve sur une plateforme open source.

Exemple de communication avec la librairie Neo4J le protocol "Bolt":

```
1 Driver driver = GraphDatabase.driver("bolt://localhost:7687",
      AuthTokens.basic("matrix", "neo"));
2 Session session = driver.session();
3 session.run("CREATE (a:Person {firstName:{name},lastName:{lastName}})",
      parameters("firstName","Gilles","lastName","Bodart"));
4 session.close();
5 driver.close();
```

Bien que cette libraire soit efficace, elle ne peut être utilisée avec un autre SGBD.

3.1.4 Librairie OrientDB

La librairie OrientDB pour java est composée de trois éléments:

- Graph API
- Document API
- Object API

La graph API d'OrientDB couvre 80% des uses cases classique d'un utilisateur, elle supporte tout les modèles de représentations comme un unique multi-modele de donnée. Elle travaille sur des "Vertex" et des "Edges", respectivement nœud et relations et est compatible avec le standard TinkerPop

La document API elle permet de couvrir les 20% restants, elle est plus simple d'utilisation mais n'offre une utilisation plus atomique, les relations ne peuvent par exemple qu'être unidirectionnelles tandis que la bidirectionnalité est offerte dans l'API précédent. Un document est un élément unique, pour gérer les relations entre les différents nœuds de notre modèle, il faudra gérer manuellement chaque connexion.

La dernière librairie se focalise sur les objets java, elle n'a plus été améliorée depuis la version 1.5 d'OrientDB[2]

3.2 Les frameworks

3.2.1 Apache TinkerPop

Apache TinkerPop est Framework open source d'utilisation de graph, il regroupe un grand ensemble de fonctionnalités et d'algorithmes. Son écosystème est de plus en plus agrémenté de librairies externes développées par de tierces personnes.

Il se considère lui-même comme étant complexe d'utilisation pour les nouveaux développeurs car il nécessite l'utilisation d'un environnement qui leur est propre, la "Gremlin-console".

Les développeurs de ce framework on mit au point un langage de travail sur les graphs appelé "Gremlin". Le langage dont nous avons parlé en section 2.1.2 s'est grandement inspirer de ce dernier.

Exemple d'utilisation

```
gremlin> g.V() /* récupère tout les noeuds du graph */
==>v[1]
==>v[2]
==>v[3]
==>v[4]
==>v[5]
==>v[6]
gremlin> g.V(1) /* récupère le noeud d'Id 1 du graph*/
==>v[1]
gremlin> g.V(1).values('name') /* récupère le noeud d'Id 1 du graph et
    renvoie sa propriété 'name' */
==>marko
```

[2] À l'heure de l'écriture de ce mémoire, nous sommes à la version 2.2 et la version 3 à bientôt fini sa phase de "bêta test"

```
gremlin> g.V(1).outE('knows') /* récupère les relations 'knows' partant du
    noeud d'Id 1 */
==>e[7][1-knows->2]
==>e[8][1-knows->4]
gremlin> g.V(1).outE('knows').inV().values('name') /* retourne le nom des
    personnes que le noeud 1 connait */
==>vadas
==>josh
gremlin> g.V(1).out('knows').values('name') /* même résultat */
==>vadas
==>josh
gremlin> g.V(1).out('knows').has('age', gt(30)).values('name') /* retourne le
    nom des personnes plus vielle que 30 ans que le noeud 1 connait */
==>josh
```

Comme expliqué plus haut, le langage doit être exécuté dans un invite de commande ou dans un serveur Gremlin, nous sommes dès lors contraint d'installer et de configurer ce dernier pour pointer vers l'un ou l'autre SGBD.

Part II

Contribution

Chapter 4

Application possibles des BDOG

4.1 L'exemple parfait

Pour la suite de ce mémoire, il est primordiale d'avoir un exemple concret de modèle habituellement persisté avec un modèle relationnelle mais qui serait sans le moindre problème transposable dans une BDOG.

L'école

Imaginons la représentation d'une école composée d'élèves, de professeurs, de classes, de salles techniques, de travailleurs, ... dont le diagramme de classes se trouve en Annexe (7.1).

Les liens entre les différents objets peuvent être fortement nombreux, nous pouvons imaginer que les étudiants sont amis, membres de famille, ennemis, amoureux, ... de même que pour les professeurs, ils peuvent avoir les mêmes relations que les étudiants mais de plus peuvent être assigné a des cours, a des classes, ... toutes ces liaisons peuvent être fortement compliquées à représenter sur une base de donnée relationnelles. L'utilisation d'une BDOG semble être tout à fait adéquate. Durant la suite de ce mémoire, nous utiliserons donc ce modèle.

4.2 Critères de comparaisons

Dans un premier temps, il est primordiale de mettre en place une liste de critères permettant de faciliter le choix de la BDOG.

4.3 Comparaison des plus grandes BDOG

BDOG / Critère	Neo4J	OrientDB
Schema de donnée strict	X	✓
Format de donnée	JSON	JSON
Principe d'héritage	X	✓
Clusturing	Configurable	Natif
Communication	BOLT (protocole propriétaire)	REST (via HTTP)
Évolutivité	L'absence de schéma strict entraîne une grande liberté dans la création et la modification des nœuds	Les classes doivent préalablement configurées sur le serveur
Rapidité[1]	7m 10s	4m 31s
Modèle de représentation	Uniquement graph	multimodèle

Une idée d'arbre de décision devrait, à la fin de cette section, permettre à un utilisateur muni de ses besoins, de choisir la BDOG la plus adaptée à son projet.

4.4 Piste de normalisation

Comme nous l'avons présenter dans le point 1 nous nous apercevons que le problème principal du NoSql est le manque de standard d'utilisation de ces SGBD, pour ce faire, nous allons tenter de lister un des objectifs de normalisations.

- Utilisation d'un langage unique
- Système de transaction
- Opération de base de persistances CRUD
- Possibilité de créer des méthodes stockée
- Contrôle sur les propriétés.

La suite de ce mémoire portant sur le développement d'un framework java, tentera d'offrir ces objectifs de manière abstraite aux développeurs.

[1] 10 000 requêtes sur le même graph de donnée avec la même requête au travers d'un appel Java

Chapter 5

Le framework

5.1 Utilisation

Ce framework à été basé sur l'utilisation d'annotations, à l'heure actuelle, il permet d'effectuer deux opérations:

5.1.1 La persistance

La persistance est l'élément primordiale de ce framework, afin de ne pas avoir de lien avec une BDOG propre, nous nous sommes basé sur le modèle orienté objet. La persistance de celui-ci ainsi que des éléments qui lui sont lié est paramétrisation dans un fichier de configuration nécessaire à l'initialisation de cet outil.

Le fichier de configuration doit se nommer "config.archipelago.json" et être présent dans les ressources du projet.

Voici le canevas de ce fichier :

```
{
  "database": {
    "type": "", // Choix entre NEO4J et ORIENT_DB
    "username": "",
    "password": "",
    "url": "",
    "name": "", // Nom de la base de donnée
    "embedded": false, // boolean Optionnel
    "port": 1234 //
  },
  "deepness": 3, // Profondeur de persistance voir ci-après
  "domainRootPackage": "org.archipelago.test.domain.school"
  // package où se trouve les classes du domaine
}
```

Afin d'assurer la persistance des éléments liés, nous avons opté pour une approche récursive en "depth-first" avec marquage de sommet, en effet, pour créer un lien entre deux éléments, il faut

absolument que ceux si soit présent dans la base de donnée.

La propriété "deepness" est primordiale pour la performance de ce framework, elle permet de définir la profondeur de persistance d'un objet. Ce procédé empêche de tomber dans des boucles infinies, courante lorsque nous travaillons avec de modèles ayant de relations bidirectionnelles comme suit :

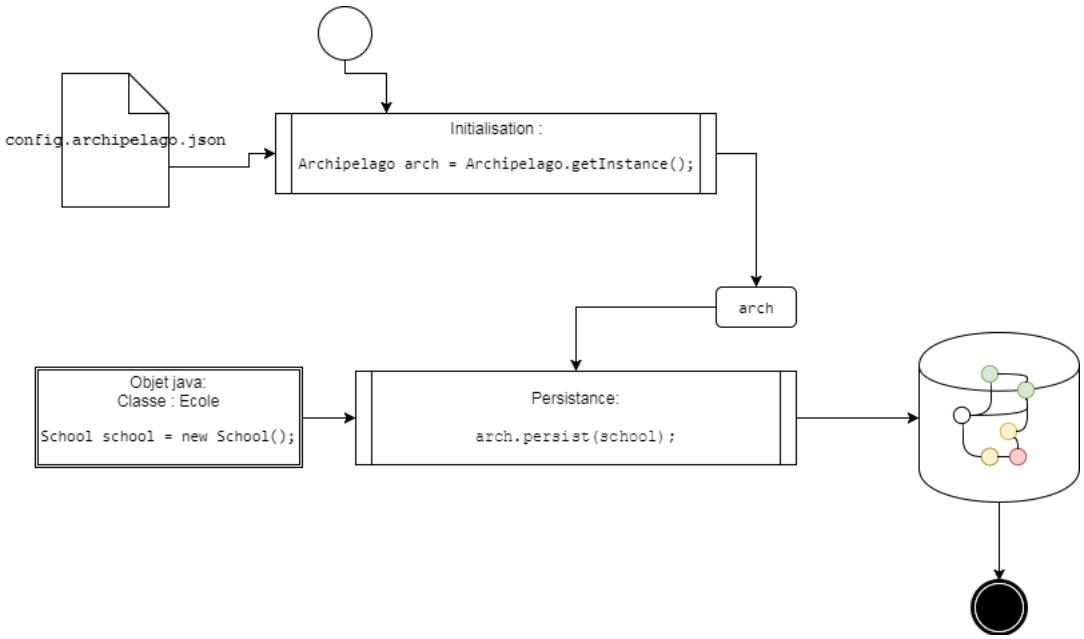
```
Archipelago arch = Archipelago.getInstance();

Student gilles = new Student();
Student antoine = new Student();

gilles.setFriend(antoine);
antoine.setFriend(gilles);

arch.persist(gilles);
```

Ainsi avec cette valeur sentinelle nous rompons la boucle de persistance apres avoir atteint la profondeur souhaité.



5.1.2 La récupération

La récupération de l'information est la raison pour laquelle nous la stockons. A quoi bon enregistrer une valeur dans une base de donnée si nous ne souhaitons jamais l'utiliser ultérieurement ?

Pour cela, Archipelago vient avec un système de création de requête. Grâce à ce procédé, que votre BDOG soit OrientDB ou Neo4J, vous serez en mesure de récupérer votre objet sans changer votre code source.

Voici un exemple:

```
Archipelago arch = Archipelago.getInstance();

```

```
ArchipelagoQuery aq = a.getQueryBuilder()
                .of(Student.class)
                .build();
List<Object> nodes = arch.execute(aq);
```

Cette requête va rechercher l'ensemble des étudiants de la base de donnée ainsi que les objets qui leurs sont liés.

Un système de condition est également mit en place, il permet d'affiner le résultat obtenu:

```
Archipelago arch = Archipelago.getInstance();

ArchipelagoQuery aq = a.getQueryBuilder()
                .of(Student.class)
                .where(of("firstName","Gilles"), ConditionQualifier.Equal)
                .build();
List<Object> nodes = arch.execute(aq);
```

Cette requête va quant à elle rechercher les étudiants ayant "Gilles" comme prénom.

Nous pouvons aussi ajouter des opérateurs logique "OU" et "ET" comme suit :

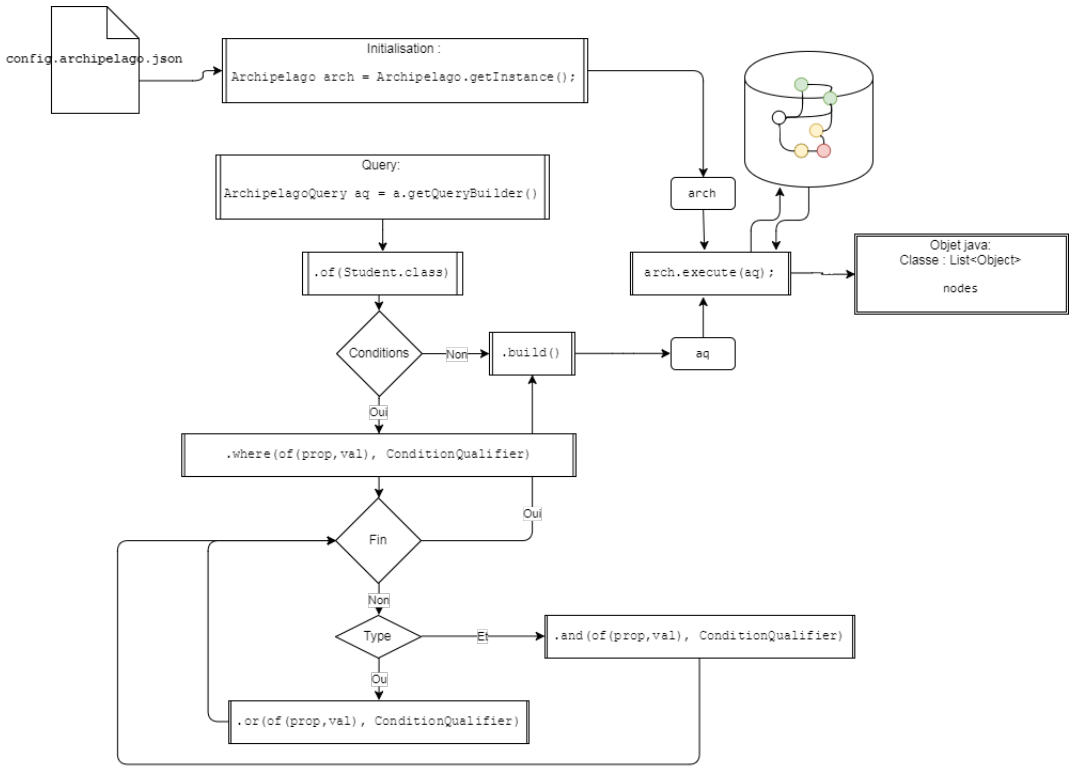
```
Archipelago arch = Archipelago.getInstance();

ArchipelagoQuery aq = a.getQueryBuilder()
                .of(Student.class)
                .where(of("lastName", "Bodart"), ConditionQualifier.EQUAL)
                .and(of("firstName", "Gilles"), ConditionQualifier.EQUAL)
                .or(of("firstName", "Thomas"), ConditionQualifier.EQUAL)
                .build();
List<Object> nodes = arch.execute(aq);
```

Le procédé mit en place à l'heure actuel ne permet pas de faire des conditions fortement complexes, il va ajouter l'élément de condition à la suite de la requête actuelle, pour l'exemple précédent, nous aurons donc :

```
lastName = "Bodart" AND ( firstName = "Gilles" OR firstName = "Thomas")
```

ce qui se retournera donc la liste des étudiants ayant "Bodart" comme nom de famille et "Gilles" ou "Thomas" comme prénom. À L'heure actuelle, il n'est cependant pas possible d'effectuer des requêtes sur les relations entre les objets, cette piste est primordiale pour une deuxième version de ce framework.



5.2 Schema conceptuel

Les schémas conceptuels représentant une modèle exemple annoté des éléments du framework sera fourni et commenté dans cette section. Cela permettra au lecteur une meilleur compréhension.

5.3 Documentation

Une documentation claire et précise sur l'utilisation du framework Archipelago sera présente dans cette section. Un ensemble entre une documentation fonctionnelle et une documentation technique faite avec JavaDoc.

5.4 Processus

Description du processus implémenté sur base d'un exemple claire. Explications des différents choix d'implémentations et de chaque étape.

Chapter 6

Evaluation

6.1 Points forts

Autocritique du framework, sur base de test qualitatif et ou quantitatif. Evaluations : usability, performence, qualité, cohérence

6.2 Points faibles

Autocritique du framework, sur base de test qualitatif et ou quantitatif. Evaluations : usability, performence, qualité, cohérence

6.3 Retour d'information

Si le temps nous le permet, une analyse des retours utilisateur sera faite en fin de mémoire.

Chapter 7

Conclusion

7.1 Piste de réflexions

Une introspection sur le projet sera expliqué dans cette section, les idées innachevés y seront décrites en tant que piste de réflexions.

7.2 Archipelago en résumé

Le mémoire sera conclu avec un explication transversale et complète du framework, permettant au lecteur de garder une bonne impression sur le nouvel outil que sera ce framework.

Part III

Annexes

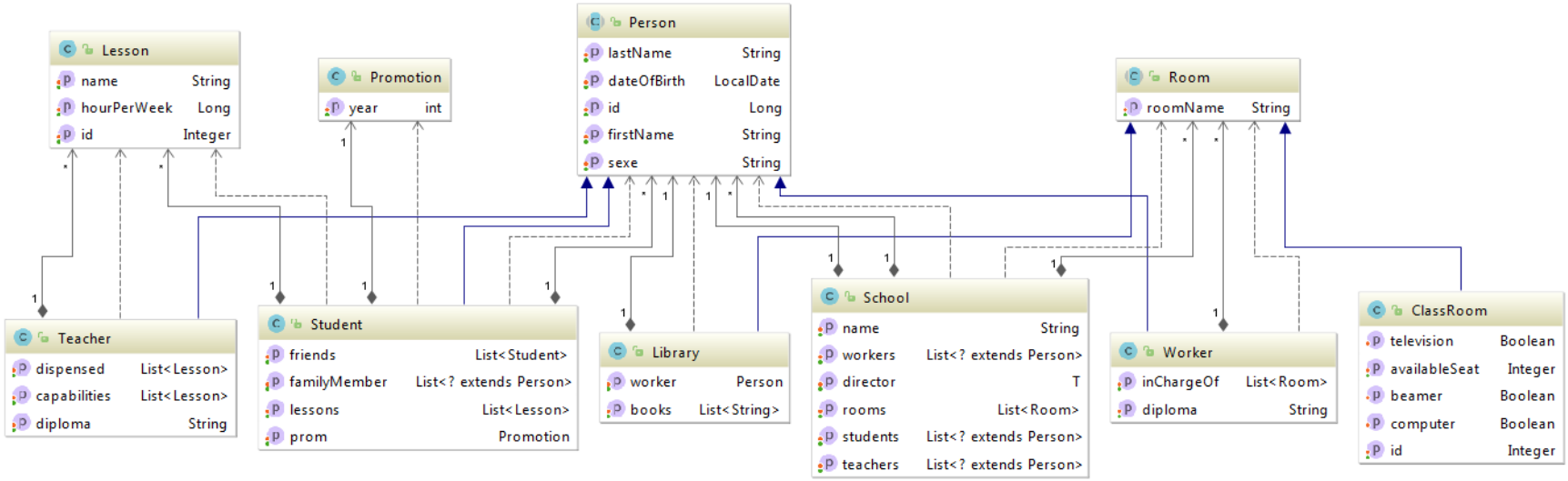


FIGURE 7.1 – Diagramme de Classe du modèle École

Lexique

SQL	Structured Query Language
NoSQL	Not only SQL
SGBD	Système de gestion de base de données
SGBDR	Système de gestion de base de données relationnelles
BDOG	Base de donnée orientée graph
JSON	JavaScript Object Notation
Vertex	Sommet
Edge	bord
Cluster	littéralement grappe, cela représente en informatique un groupe d'élément
API	Application Programming Interface
REST	Representational state transfer, style d'architecture pour les systèmes hypermédia distribués

Code Sources

Bridge.java

```java
package org.archipelago.core.annotations;

import java.lang.annotation.*;

/**
 * Created by Gilles Bodart on 18/08/2016.
 */
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Bridge {


    String descriptor();

    boolean biDirectionnal() default false;

}
```

ArchipelagoQuery.java

```java
package org.archipelago.core.builder;

import java.util.List;

/**
 * Created by Gilles Bodart on 19/07/2017.
 */
public class ArchipelagoQuery {

    private final String query;
    private final List<String> keys;
    private final Class<?> target;

    private final boolean relation;
    private final Object from;
    private final Object to;
    private final Object descriptor;
    private final boolean biDirectionnal;

    public ArchipelagoQuery(String query, List<String> keys, Class<?> target,
        boolean relation, Object from, Object to, Object descriptor, boolean
        biDirectionnal) {
        this.query = query;
        this.keys = keys;
        this.target = target;
        this.from = from;
        this.to = to;
        this.relation = relation;
```

```
27          this.descriptor = descriptor;
28          this.biDirectionnal = biDirectionnal;
29      }
30
31      public String getQuery() {
32          return query;
33      }
34
35      public List<String> getKeys() {
36          return keys;
37      }
38
39      public Class<?> getTarget() {
40          return target;
41      }
42
43      public boolean isRelation() {
44          return relation;
45      }
46
47      public Object getFrom() {
48          return from;
49      }
50
51      public Object getTo() {
52          return to;
53      }
54
55      public Object getDescriptor() {
56          return descriptor;
57      }
58
59      public boolean isBiDirectionnal() {
60          return biDirectionnal;
61      }
62
63      @Override
64      public int hashCode() {
65          return super.hashCode();
66      }
67  }
```

ArchipelagoScriptBuilder.java

```
1   package org.archipelago.core.builder;
2
3   import com.google.common.collect.Lists;
4   import org.apache.logging.log4j.LogManager;
5   import org.apache.logging.log4j.Logger;
6   import org.archipelago.core.annotations.Bridge;
7   import org.archipelago.core.domain.DescriptorWrapper;
```

```
8  import org.archipelago.core.util.ArchipelagoUtils;
9
10 import java.lang.reflect.Field;
11 import java.util.List;
12 import java.util.Set;
13
14 /**
15  * @author Gilles Bodart
16  */
17 public abstract class ArchipelagoScriptBuilder {
18
19     public static final String TEMPLATE_ROOT_PATH = "StringTemplate";
20     protected final static Logger LOGGER =
           LogManager.getLogger(ArchipelagoScriptBuilder.class);
21
22
23     public abstract String makeCreate(Object object);
24
25     public List<Object> fillCreate(Object object) {
26         final List<Object> parameters = Lists.newArrayList();
27         Class<?> clazz = object.getClass();
28         Set<Field> fields = ArchipelagoUtils.getAllFields(clazz);
29         if (String.class.equals(clazz)) {
30             parameters.add("name");
31             parameters.add(object);
32         } else {
33             for (Field field : fields) {
34                 if (!field.isAnnotationPresent(Bridge.class)) {
35                     Object prop = ArchipelagoUtils.get(clazz, field, object);
36                     if (null != prop) {
37                         parameters.add(field.getName());
38                         parameters.add(ArchipelagoUtils.formatQueryValue(prop,
                               true));
39                     }
40                 }
41             }
42         }
43
44         return parameters;
45     }
46
47     public String makeMatch(Object object) {
48         throw new UnsupportedOperationException("Not implemented yet");
49     }
50
51     public String makeMatch(Object object, boolean allObject) {
52         throw new UnsupportedOperationException("Not implemented yet");
53     }
54
55     public String makeRelation(int idA, int idB, String name, Class<?>
```

```
        descriptor) {
56          throw new UnsupportedOperationException("Not implemented yet");
57      }
58
59      public String makeRelation(int idA, int idB, String name, List<String>
            props) {
60          throw new UnsupportedOperationException("Not implemented yet");
61      }
62
63      public String makeRelation(Object idA, Object idB, DescriptorWrapper
            description) {
64          throw new UnsupportedOperationException("Not implemented yet");
65      }
66
67      public String makeRelation(Object idA, Object idB, String description) {
68          throw new UnsupportedOperationException("Not implemented yet");
69      }
70  }
```

ConditionQualifier.java

```
1   package org.archipelago.core.builder;
2
3   /**
4    * Created by Gilles Bodart on 19/07/2017.
5    */
6   public enum ConditionQualifier {
7       EQUAL("="), NOT_EQUAL("!="), LESS_THAN("<="), MORE_THAN(">="),
            STRICT_LESS_THAN("<"), STRICT_MORE_THAN(">");
8
9       private String symbol;
10
11      private ConditionQualifier(String symbol) {
12          this.symbol = symbol;
13      }
14
15      public String getSymbol() {
16          return symbol;
17      }
18  }
```

Neo4JBuilder.java

```
1   package org.archipelago.core.builder;
2
3   import org.archipelago.core.annotations.Bridge;
4   import org.archipelago.core.util.ArchipelagoUtils;
5   import org.archipelago.core.util.StringTemplateFactory;
6   import org.stringtemplate.v4.ST;
7   import org.stringtemplate.v4.STGroup;
8
```

```java
import java.lang.reflect.Field;
import java.util.List;

/**
 * @author Gilles Bodart
 */
public class Neo4JBuilder extends ArchipelagoScriptBuilder {

    private static final String NEO4J_FOLDER = "\\Neo4J";

    public String makeCreate(Object object) {
        Class<?> clazz = object.getClass();
        final String nodeTemplatePath = TEMPLATE_ROOT_PATH + NEO4J_FOLDER +
            "\\node.stg";
        final STGroup group =
            StringTemplateFactory.buildSTGroup(nodeTemplatePath);
        final ST st = group.getInstanceOf("ClassNeo4J");
        st.add("clazz", clazz);
        if (String.class.equals(clazz)) {
            st.add("props", "name");
        } else {
            for (Field field : ArchipelagoUtils.getAllFields(clazz)) {
                if (!field.isAnnotationPresent(Bridge.class)) {
                    if (null != ArchipelagoUtils.get(clazz, field, object)) {
                        st.add("props", field.getName());
                    }
                }
            }
        }
        String create = st.render();
        LOGGER.debug(String.format("CREATE for class %s : [%s]",
            clazz.getSimpleName(), create));
        return create;
    }


    public String makeMatch(Object object) {
        return makeMatch(object, true);
    }

    public String makeMatch(Object object, boolean allObject) {
        Class<?> clazz = object.getClass();
        final String nodeTemplatePath = String.format("%s/%s/%s%s",
            TEMPLATE_ROOT_PATH, NEO4J_FOLDER, "\\match", allObject ? ".stg" :
            "Id.stg");
        final STGroup group =
            StringTemplateFactory.buildSTGroup(nodeTemplatePath);
        final ST st = group.getInstanceOf("MatchNeo4J");
        st.add("clazz", clazz);
        if (String.class.equals(clazz)) {
```

```
53              st.add("props", "name");
54          } else {
55              for (Field field : ArchipelagoUtils.getAllFields(clazz)) {
56                  if (!field.isAnnotationPresent(Bridge.class)) {
57                      if (null != ArchipelagoUtils.get(clazz, field, object)) {
58                          st.add("props", field.getName());
59                      }
60                  }
61              }
62          }
63          String match = st.render();
64          LOGGER.debug(String.format("MATCH for class %s : [%s]",
                clazz.getSimpleName(), match));
65          return match;
66      }
67
68      public String makeRelation(int idA, int idB, String name) {
69          final String nodeTemplatePath = String.format("%s/%s/%s",
                TEMPLATE_ROOT_PATH, NEO4J_FOLDER, "\\relation.stg");
70          final STGroup group =
                StringTemplateFactory.buildSTGroup(nodeTemplatePath);
71          final ST st = group.getInstanceOf("RelationNeo4J");
72          st.add("idA", idA);
73          st.add("idB", idB);
74          st.add("name", name);
75          String relationQuery = st.render();
76          LOGGER.debug(String.format("CREATE Relation from %d to %d : [%s]",
                idA, idB, relationQuery));
77          return relationQuery;
78      }
79
80      public String makeRelation(int idA, int idB, String name, Class<?>
            descriptor) {
81
82          final String nodeTemplatePath = String.format("%s/%s/%s",
                TEMPLATE_ROOT_PATH, NEO4J_FOLDER, "\\relationWithProp.stg");
83          final STGroup group =
                StringTemplateFactory.buildSTGroup(nodeTemplatePath);
84          final ST st = group.getInstanceOf("RelationNeo4J");
85          st.add("idA", idA);
86          st.add("idB", idB);
87          st.add("name", name);
88          for (Field field : ArchipelagoUtils.getAllFields(descriptor)) {
89              st.add("properties", field.getName());
90          }
91          String relationQuery = st.render();
92          LOGGER.debug(String.format("CREATE Relation from %d to %d : [%s]",
                idA, idB, relationQuery));
93          return relationQuery;
94      }
```

```
95
96      public String makeRelation(int idA, int idB, String name, List<String>
            props) {
97          final String nodeTemplatePath = String.format("%s/%s/%s",
                TEMPLATE_ROOT_PATH, NEO4J_FOLDER, "\\relationWithProp.stg");
98          final STGroup group =
                StringTemplateFactory.buildSTGroup(nodeTemplatePath);
99          final ST st = group.getInstanceOf("RelationNeo4J");
100         st.add("idA", idA);
101         st.add("idB", idB);
102         st.add("name", name);
103         for (String prop : props) {
104             st.add("properties", prop);
105         }
106         String relationQuery = st.render();
107         LOGGER.debug(String.format("CREATE Relation from %d to %d : [%s]",
                idA, idB, relationQuery));
108         return relationQuery;
109     }
110
111 }
```

Neo4JQueryImpl.java

```
1  package org.archipelago.core.builder;
2
3  import org.archipelago.core.util.ArchipelagoUtils;
4
5  import java.util.ArrayList;
6  import java.util.List;
7
8  /**
9   * Created by Gilles Bodart on 19/07/2017.
10  */
11 public class Neo4JQueryImpl extends QueryBuilder {
12
13     private StringBuilder pending = new StringBuilder();
14     private Class<?> target;
15
16     private List<String> elementToReturn = new ArrayList<>();
17
18     public static QueryBuilder init() {
19         Neo4JQueryImpl impl = new Neo4JQueryImpl();
20         impl.pending.append("MATCH ");
21         return impl;
22     }
23
24     @Override
25     public QueryBuilder of(Class<?> clazz) {
26         pending.append(String.format("p=(n:%s)-[rel]-() ",
               clazz.getSimpleName()));
```

```
27          this.target = clazz;
28          ArchipelagoUtils.getAllFields(clazz).stream().forEach(field ->
                elementToReturn.add(field.getName()));
29          return this;
30      }
31
32      @Override
33      public QueryBuilder where(QueryElement element, ConditionQualifier
            conditionQualifier) {
34          condition(element, conditionQualifier, "WHERE");
35          return this;
36      }
37
38      @Override
39      public QueryBuilder and(QueryElement element, ConditionQualifier
            conditionQualifier) {
40          condition(element, conditionQualifier, "AND");
41          return this;
42      }
43
44      @Override
45      public QueryBuilder or(QueryElement element, ConditionQualifier
            conditionQualifier) {
46          condition(element, conditionQualifier, "OR");
47          return this;
48      }
49
50      private void condition(QueryElement element, ConditionQualifier
            conditionQualifier, String logicSym) {
51          Object formatedValue =
                ArchipelagoUtils.formatQueryValue(element.getValue());
52          pending.append(String.format(" %s %s %s %s",
53                  logicSym,
54                  "n." + element.getKey(),
55                  conditionQualifier.getSymbol(),
56                  element.getValue() instanceof String ? "\"" + formatedValue +
                        "\"" : formatedValue));
57      }
58
59      @Override
60      public ArchipelagoQuery build() {
61          pending.append(" RETURN p");
62          return new ArchipelagoQuery(pending.toString(), elementToReturn,
                target, relation, from, to, descriptor, biDirectionnal);
63      }
64  }
```

OrientDBBuilder.java

```
1   package org.archipelago.core.builder;
2
```

```
3  import org.archipelago.core.annotations.Bridge;
4  import org.archipelago.core.domain.DescriptorWrapper;
5  import org.archipelago.core.util.ArchipelagoUtils;
6  import org.archipelago.core.util.StringTemplateFactory;
7  import org.stringtemplate.v4.ST;
8  import org.stringtemplate.v4.STGroup;
9
10 import java.lang.reflect.Field;
11 import java.util.Iterator;
12
13 /**
14  * @author Gilles Bodart
15  */
16 public class OrientDBBuilder extends ArchipelagoScriptBuilder {
17
18     private static final String ORIENT_FOLDER = "\\OrientDB";
19
20     @Override
21     public String makeCreate(Object object) {
22         final StringBuilder sb = new StringBuilder();
23         sb.append(String.format("{\"@class\":\"%s\"",
               object.getClass().getSimpleName()));
24         Iterator<Object> iterator = fillCreate(object).iterator();
25         while (iterator.hasNext()) {
26             sb.append(",");
27             sb.append(String.format("\"%s\":%s", iterator.next(),
                   iterator.next()));
28         }
29         sb.append("}");
30         LOGGER.debug(String.format("CREATE for class %s : [%s]",
               object.getClass().getSimpleName(), sb.toString()));
31         return sb.toString();
32     }
33
34     @Override
35     public String makeRelation(Object idA, Object idB, DescriptorWrapper
           description) {
36         return String.format("CREATE EDGE %s FROM %s TO %s SET created = %s",
37                 description.getName(), idA, idB,
                       ArchipelagoUtils.formatQueryValue(description.getCreated(),
                       true));
38     }
39
40     public String makeMatch(Object object) {
41         Class<?> clazz = object.getClass();
42         final String nodeTemplatePath = String.format("%s/%s/%s",
               TEMPLATE_ROOT_PATH, ORIENT_FOLDER, "\\match.stg");
43         final STGroup group =
               StringTemplateFactory.buildSTGroup(nodeTemplatePath);
44         final ST st = group.getInstanceOf("MatchOrientDB");
```

```
        st.add("clazz", clazz);
        if (String.class.equals(clazz)) {
            st.add("props", "name");
        } else {
            for (Field field : ArchipelagoUtils.getAllFields(clazz)) {
                if (!field.isAnnotationPresent(Bridge.class)) {
                    Object o = ArchipelagoUtils.get(clazz, field, object);
                    if (null != o) {
                        st.add("props", new ValueWrapper(field.getName(),
                            ArchipelagoUtils.formatQueryValue(o, true, true)));
                    }
                }
            }
        }
        String match = st.render();
        LOGGER.debug(String.format("MATCH for class %s : [%s]",
            clazz.getSimpleName(), match));
        return match;
    }

    private class ValueWrapper {
        private String name;
        private Object value;

        public ValueWrapper(String name, Object value) {
            this.name = name;
            this.value = value;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public Object getValue() {
            return value;
        }

        public void setValue(Object value) {
            this.value = value;
        }
    }
}
```

OrientDBQueryImpl.java

```
package org.archipelago.core.builder;

```

```java
import org.archipelago.core.util.ArchipelagoUtils;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by Gilles Bodart on 19/07/2017.
 */
public class OrientDBQueryImpl extends QueryBuilder {

    private StringBuilder pending = new StringBuilder();
    private Class<?> target;

    private List<String> elementToReturn = new ArrayList<>();

    public static QueryBuilder init() {
        OrientDBQueryImpl impl = new OrientDBQueryImpl();
        impl.pending.append("SELECT ");
        return impl;
    }

    @Override
    public QueryBuilder of(Class<?> clazz) {
        pending.append(String.format(" FROM %s ", clazz.getSimpleName()));
        this.target = clazz;
        ArchipelagoUtils.getAllFields(clazz).stream().forEach(field ->
            elementToReturn.add(field.getName()));
        return this;
    }


    @Override
    public QueryBuilder where(QueryElement element, ConditionQualifier
         conditionQualifier) {
        condition(element, conditionQualifier, "WHERE");
        return this;
    }

    @Override
    public QueryBuilder and(QueryElement element, ConditionQualifier
         conditionQualifier) {
        condition(element, conditionQualifier, "AND");
        return this;
    }

    @Override
    public QueryBuilder or(QueryElement element, ConditionQualifier
         conditionQualifier) {
        condition(element, conditionQualifier, "OR");
        return this;
```

```
    }

    private void condition(QueryElement element, ConditionQualifier
        conditionQualifier, String logicSym) {

        pending.append(String.format(" %s %s %s %s",
                logicSym,
                "" + element.getKey(),
                conditionQualifier.getSymbol(),
                ArchipelagoUtils.formatQueryValue(element.getValue(), true,
                    true)));
    }

    @Override
    public ArchipelagoQuery build() {
        return new ArchipelagoQuery(pending.toString(), elementToReturn,
            target, relation, from, to, descriptor, biDirectionnal);
    }
}
```

QueryBuilder.java

```
package org.archipelago.core.builder;

/**
 * Created by Gilles Bodart on 19/07/2017.
 */
public abstract class QueryBuilder {


    protected boolean relation = false;
    protected Object from;
    protected Object to;
    protected Object descriptor;
    protected boolean biDirectionnal = false;

    public static QueryBuilder init() {
        throw new UnsupportedOperationException("QueryBuilder is abstract and
            can not be implemented");
    }


    public QueryBuilder linkObjects() {
        this.relation = true;
        return this;
    }

    public QueryBuilder from(Object from) {
        this.from = from;
        return this;
    }
```

```
29
30      public QueryBuilder to(Object to) {
31          this.to = to;
32          return this;
33      }
34
35      public QueryBuilder biDirectionnal() {
36          this.biDirectionnal = biDirectionnal;
37          return this;
38      }
39
40      public QueryBuilder with(Object descriptor) {
41          this.descriptor = descriptor;
42          return this;
43      }
44
45      public abstract QueryBuilder of(Class<?> element);
46
47      public abstract QueryBuilder where(QueryElement element,
            ConditionQualifier conditionQualifier);
48
49      public abstract QueryBuilder and(QueryElement element, ConditionQualifier
            conditionQualifier);
50
51      public abstract QueryBuilder or(QueryElement element, ConditionQualifier
            conditionQualifier);
52
53      public abstract ArchipelagoQuery build();
54  }
```

QueryElement.java

```
1   package org.archipelago.core.builder;
2
3   /**
4    * Created by Gilles Bodart on 19/07/2017.
5    */
6   public class QueryElement {
7       private final String key;
8       private final Object value;
9
10      private QueryElement(String key, Object value, Boolean id) {
11          this.key = key;
12          this.value = value;
13      }
14
15      public static QueryElement qualifier(String qualifier) {
16          QueryElement qe = new QueryElement(qualifier, "", false);
17          return qe;
18      }
19
```

```
    public static QueryElement of(String key, Object value) {
        QueryElement qe = new QueryElement(key, value, false);
        return qe;
    }

    public String getKey() {
        return key;
    }

    public Object getValue() {
        return value;
    }

}
```

ArchipelagoConfig.java

```
package org.archipelago.core.configurator;

import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Created by ABM589 on 18/07/2017.
 */
public class ArchipelagoConfig {

    @JsonProperty()
    private DatabaseConfig database;

    @JsonProperty
    private Integer deepness;

    @JsonProperty
    private String domainRootPackage;

    public ArchipelagoConfig() {
    }

    public DatabaseConfig getDatabase() {
        return database;
    }

    public void setDatabase(DatabaseConfig database) {
        this.database = database;
    }

    public Integer getDeepness() {
        return deepness;
    }

    public void setDeepness(Integer deepness) {
```

```
        this.deepness = deepness;
    }

    public String getDomainRootPackage() {
        return domainRootPackage;
    }

    public void setDomainRootPackage(String domainRootPackage) {
        this.domainRootPackage = domainRootPackage;
    }
}
```

DatabaseConfig.java

```
package org.archipelago.core.configurator;

import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Created by ABM589 on 18/07/2017.
 */
public class DatabaseConfig {

    @JsonProperty
    private DatabaseType type;

    @JsonProperty
    private String username;

    @JsonProperty
    private String password;

    @JsonProperty
    private String url;

    @JsonProperty
    private int port;

    @JsonProperty
    private Boolean embedded;

    @JsonProperty
    private String name;

    public DatabaseConfig() {
    }

    public DatabaseType getType() {
        return type;
    }

```

```
38      public void setType(DatabaseType type) {
39          this.type = type;
40      }
41
42      public String getUsername() {
43          return username;
44      }
45
46      public void setUsername(String username) {
47          this.username = username;
48      }
49
50      public String getPassword() {
51          return password;
52      }
53
54      public void setPassword(String password) {
55          this.password = password;
56      }
57
58      public String getUrl() {
59          return url;
60      }
61
62      public void setUrl(String url) {
63          this.url = url;
64      }
65
66      public Boolean getEmbedded() {
67          return embedded;
68      }
69
70      public void setEmbedded(Boolean embedded) {
71          this.embedded = embedded;
72      }
73
74      public int getPort() {
75          return port;
76      }
77
78      public void setPort(int port) {
79          this.port = port;
80      }
81
82      public String getName() {
83          return name;
84      }
85
86      public void setName(String name) {
87          this.name = name;
```

```
88      }
89  }
```

DatabaseType.java

```
1   package org.archipelago.core.configurator;
2
3   /**
4    * Created by Gilles Bodart on 18/07/2017.
5    */
6   public enum DatabaseType {
7
8       NEO4J, ORIENT_DB, RELATIONNAL;
9   }
```

DescriptorWrapper.java

```
1   package org.archipelago.core.domain;
2
3   import org.archipelago.core.annotations.Bridge;
4
5   import java.time.LocalDate;
6
7   /**
8    * Created by Gilles Bodart on 23/07/2017.
9    */
10  public class DescriptorWrapper {
11
12      @Bridge(descriptor = "")
13      private String name;
14      private LocalDate created;
15
16      public DescriptorWrapper() {
17          this.created = LocalDate.now();
18      }
19
20      public DescriptorWrapper(String descriptorName) {
21          this();
22          this.name = descriptorName;
23      }
24
25      public String getName() {
26          return name;
27      }
28
29      public void setName(String name) {
30          this.name = name;
31      }
32
33      public LocalDate getCreated() {
34          return created;
```

```
35      }
36
37      public void setCreated(LocalDate created) {
38          this.created = created;
39      }
40  }
```

OrientDBCluster.java

```
1   package org.archipelago.core.domain;
2
3   /**
4    *
5    * @author Gilles Bodart
6    *
7    */
8   public class OrientDBCluster {
9
10      private Integer clusterAmount;
11      private String clusterName;
12
13      public OrientDBCluster() {
14
15      }
16
17      public OrientDBCluster(final Integer clusterAmount, final String
            clusterName) {
18          super();
19          this.clusterAmount = clusterAmount;
20          this.clusterName = clusterName;
21      }
22
23      public Integer getClusterAmount() {
24          return clusterAmount;
25      }
26
27      public void setClusterAmount(final Integer clusterAmount) {
28          this.clusterAmount = clusterAmount;
29      }
30
31      public String getClusterName() {
32          return clusterName;
33      }
34
35      public void setClusterName(final String clusterName) {
36          this.clusterName = clusterName;
37      }
38
39  }
```

OrientDBErrorWrapper.java

```java
package org.archipelago.core.domain;

import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.util.List;
import java.util.Map;

/**
 * Created by Gilles Bodart on 21/07/2017.
 */
public class OrientDBErrorWrapper {

    @JsonProperty
    private List<Map<String, Object>> errors;


    public List<Map<String, Object>> getErrors() {
        return errors;
    }

    public void setErrors(List<Map<String, Object>> errors) {
        this.errors = errors;
    }

    @Override
    public String toString() {
        String ret = "";
        try {
            ret = new ObjectMapper().writeValueAsString(this);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
        return ret;
    }
}
```

OrientDBRelationWrapper.java

```java
package org.archipelago.core.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Created by Gilles Bodart on 21/07/2017.
 */
@JsonIgnoreProperties(ignoreUnknown = true)
```

```
10 public class OrientDBRelationWrapper {
11
12     @JsonProperty("@type")
13     private String type;
14     @JsonProperty("@rid")
15     private String id;
16     @JsonProperty("@version")
17     private String version;
18     @JsonProperty("in")
19     private String in;
20     @JsonProperty("out")
21     private String out;
22
23     public String getType() {
24         return type;
25     }
26
27     public void setType(String type) {
28         this.type = type;
29     }
30
31     public String getId() {
32         return id;
33     }
34
35     public void setId(String id) {
36         this.id = id;
37     }
38
39     public String getVersion() {
40         return version;
41     }
42
43     public void setVersion(String version) {
44         this.version = version;
45     }
46
47     public String getIn() {
48         return in;
49     }
50
51     public void setIn(String in) {
52         this.in = in;
53     }
54
55     public String getOut() {
56         return out;
57     }
58
59     public void setOut(String out) {
```

```
        this.out = out;
    }
}
```

OrientDBResponseWrapper.java

```
package org.archipelago.core.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Created by Gilles Bodart on 21/07/2017.
 */
@JsonIgnoreProperties(ignoreUnknown = true)
public class OrientDBResponseWrapper {

    @JsonProperty("@type")
    private String type;
    @JsonProperty("@rid")
    private String id;
    @JsonProperty("@version")
    private String version;
    @JsonProperty("@class")
    private String className;

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }

    public String getClassName() {
```

```
        return className;
    }

    public void setClassName(String className) {
        this.className = className;
    }
}
```

OrientDBResultWrapper.java

```
package org.archipelago.core.domain;

import com.fasterxml.jackson.annotation.JsonProperty;

import java.util.List;
import java.util.Map;

/**
 * Created by Gilles Bodart on 21/07/2017.
 */
public class OrientDBResultWrapper {

    @JsonProperty
    private List<Map<String, Object>> result;

    public OrientDBResultWrapper() {
        this.result = result;
    }

    public List<Map<String, Object>> getResult() {
        return result;
    }

    public void setResult(List<Map<String, Object>> result) {
        this.result = result;
    }
}
```

RelationWrapper.java

```
package org.archipelago.core.domain;

import org.apache.commons.lang3.builder.ToStringBuilder;

/**
 * Created by Gilles Bodart on 23/07/2017.
 */
public class RelationWrapper {

    private String name;
    private Object to;
```

```
12      private boolean biDirectionnal;
13
14      public Object getTo() {
15          return to;
16      }
17
18      public void setTo(Object to) {
19          this.to = to;
20      }
21
22      public String getName() {
23          return name;
24      }
25
26      public void setName(String name) {
27          this.name = name;
28      }
29
30      public boolean isBiDirectionnal() {
31          return biDirectionnal;
32      }
33
34      public void setBiDirectionnal(boolean biDirectionnal) {
35          this.biDirectionnal = biDirectionnal;
36      }
37
38      @Override
39      public String toString() {
40          return new ToStringBuilder(this)
41                  .append("name", name)
42                  .append("to", to)
43                  .append("biDirectionnal", biDirectionnal)
44                  .toString();
45      }
46  }
```

CheckException.java

```
1   package org.archipelago.core.exception;
2
3   /**
4    * Created by Gilles Bodart on 18/08/2016.
5    */
6   public class CheckException extends Exception {
7       /**
8        *
9        */
10      private static final long serialVersionUID = 6666496705775508281L;
11
12      public CheckException() {
13          super();
```

```
    }

    public CheckException(final String message) {
        super(message);
    }

    public CheckException(final String message, final Throwable cause) {
        super(message, cause);
    }

    public CheckException(final Throwable cause) {
        super(cause);
    }

    protected CheckException(final String message, final Throwable cause,
         final boolean enableSuppression, final boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

Archipelago.java

```
package org.archipelago.core.framework;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.commons.lang3.StringUtils;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.archipelago.core.builder.*;
import org.archipelago.core.configurator.ArchipelagoConfig;
import org.archipelago.core.configurator.DatabaseConfig;
import org.archipelago.core.configurator.DatabaseType;
import org.archipelago.core.domain.*;
import org.archipelago.core.exception.CheckException;
import org.archipelago.core.util.ArchipelagoUtils;
import org.glassfish.jersey.client.authentication.HttpAuthenticationFeature;
import org.neo4j.driver.v1.*;
import org.neo4j.driver.v1.types.Node;
import org.neo4j.driver.v1.types.Path;
import org.neo4j.driver.v1.types.Relationship;
import org.reflections.ReflectionUtils;
import org.reflections.Reflections;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.io.File;
import java.io.IOException;
```

```
30 import java.io.UnsupportedEncodingException;
31 import java.lang.reflect.Field;
32 import java.lang.reflect.Method;
33 import java.lang.reflect.Modifier;
34 import java.net.URLEncoder;
35 import java.util.*;
36
37 import static org.neo4j.driver.v1.Values.parameters;
38 import static org.reflections.ReflectionUtils.withModifier;
39 import static org.reflections.ReflectionUtils.withPrefix;
40
41 /**
42  * @author Gilles Bodart
43  */
44 public class Archipelago implements AutoCloseable {
45
46     public static final String CONFIG_LOCATION = "config.archipelago.json";
47     public static final String ARCHIPELAGO_ID = "ArchipelagoId";
48
49     public static final Logger LOGGER = LogManager.getLogger();
50
51     private static Archipelago instance;
52
53     private final ArchipelagoConfig archipelagoConfig;
54     private final ObjectMapper OM = new ObjectMapper();
55     private DatabaseType databaseType;
56     private Driver driver;
57     private ArchipelagoScriptBuilder builder;
58     private Client jerseyClient;
59     private WebTarget rootTarget;
60     private Reflections reflections;
61     private Map<Object, Object> sessionObject = new HashMap<>();
62
63     private Archipelago() throws CheckException {
64         // Initialization of the configuration
65         this.archipelagoConfig = getConfig();
66         DatabaseConfig dc = archipelagoConfig.getDatabase();
67         // Personalisation of the connection
68
69         this.databaseType = dc.getType();
70         this.reflections = new
              Reflections(archipelagoConfig.getDomainRootPackage());
71         switch (dc.getType()) {
72             case NEO4J:
73                 if (!dc.getEmbedded()) {
74                     String url = String.format("bolt://%s:%d", dc.getUrl(),
                          dc.getPort());
75                     LOGGER.info("Connecting to Neo4J remote server on " + url);
76                     this.driver = GraphDatabase.driver(url,
                          AuthTokens.basic(dc.getUsername(), dc.getPassword()));
```

```
77                  LOGGER.info("Connected to Neo4J remote server");
78                  this.builder = new Neo4JBuilder();
79              } else {
80                  //TODO EMBEDDED Neo4J
81              }
82              break;
83          case ORIENT_DB:
84              HttpAuthenticationFeature feature =
                    HttpAuthenticationFeature.basic(dc.getUsername(),
                    dc.getPassword());
85              this.jerseyClient = ClientBuilder.newClient();
86              this.jerseyClient.register(feature);
87              String url = String.format("http://%s:%d/", dc.getUrl(),
                    dc.getPort());
88              LOGGER.info("Connecting to Orient remote server on " + url);
89              this.rootTarget = jerseyClient.target(url);
90              this.builder = new OrientDBBuilder();
91              Response response =
                    this.rootTarget.path(String.format("connect/%s",
                    dc.getName())).request(MediaType.APPLICATION_JSON).get();
92              if (response.getStatus() != 204) {
93                  throw new CheckException("Unable to connect to Orient
                        Remote server");
94              } else {
95                  LOGGER.info("Connected to Orient remote server");
96              }
97              break;
98          default:
99              throw new CheckException("Only NEO4J and ORIENT_DB are
                    supported for the moment!");
100     }
101 }
102
103 public static Archipelago getInstance() throws CheckException {
104     if (null == instance) {
105         instance = new Archipelago();
106     }
107     return instance;
108 }
109
110 public QueryBuilder getQueryBuilder() throws CheckException {
111     switch (databaseType) {
112         case NEO4J:
113             return Neo4JQueryImpl.init();
114         case ORIENT_DB:
115             return OrientDBQueryImpl.init();
116         default:
117             throw new CheckException("Only NEO4J and ORIENT_DB are
                    supported for the moment!");
118
```

```
119         }
120     }
121
122     private ArchipelagoConfig getConfig() throws CheckException {
123         ArchipelagoConfig ac = null;
124         ObjectMapper om = new ObjectMapper();
125         try {
126             ac = om.readValue(new
                    File(getClass().getClassLoader().getResource(CONFIG_LOCATION).getFile()),
127                    ArchipelagoConfig.class);
128         } catch (IOException e) {
129             LOGGER.error(e.getMessage(), e);
130             throw new CheckException("The config file have not been found,
                    please have a 'config.archipelago.json' in your resource
                    folder", e);
131         }
132         return ac;
133     }
134
135     /**
136      * Persist an object into the database
137      *
138      * @param object the object to persist
139      * @return the Id of the persisted object
140      * @throws CheckException Object can't be null;
141      */
142     public void persist(Object object) throws CheckException {
143         persist(object, 0);
144     }
145
146     /**
147      * Persist an object into the database
148      *
149      * @param objects the objects to persist
150      * @return the Id of the persisted object
151      * @throws CheckException Object can't be null;
152      */
153     public void persist(List<Object> objects) throws CheckException {
154         for (Object object : objects) {
155             persist(object, 0);
156         }
157     }
158
159     /**
160      * Flush the objects stored in the session
161      *
162      * @return the amount of flushed elements
163      */
164     public int flushSessionObject() {
165         int size = sessionObject.size();
```

```
166         sessionObject.clear();
167         return size;
168     }
169 
170     /**
171      * Persist an object in the database
172      *
173      * @param object the object to persist
174      * @param deepness current deepness of the introspection
175      * @return the internal Id of the object (null in case or error)
176      */
177     private void persist(Object object, Integer deepness) throws
            CheckException {
178         //TODO Transaction
179         switch (databaseType) {
180             case NEO4J:
181                 Object[] params = builder.fillCreate(object).toArray();
182                 try (Session s = this.driver.session()) {
183                     s.run(builder.makeCreate(object),
                            parameters(builder.fillCreate(object).toArray()));
184                     //TODO Duplicate with scanForLink
185                     if (deepness < archipelagoConfig.getDeepness()) {
186                         List<RelationWrapper> relations =
                                ArchipelagoUtils.getChilds(object);
187                         relations.stream().forEach((relationWrapper -> {
188                             Object to = relationWrapper.getTo();
189                             if (null == getId(to)) {
190                                 try {
191                                     persist(to, deepness + 1);
192                                 } catch (CheckException e) {
193                                     LOGGER.error(e.getMessage(), e);
194                                 }
195                             } else {
196                                 scanForLink(to, deepness + 1);
197                             }
198                             link(object, relationWrapper.getTo(),
                                    relationWrapper.getName(),
                                    relationWrapper.isBiDirectionnal());
199                         }));
200                     }
201                 } catch (Exception e) {
202                     LOGGER.debug(String.format("Param [%s]",
                            StringUtils.join(params, ';')));
203                     throw e;
204                 }
205                 break;
206             case ORIENT_DB:
207                 if (null == getId(object)) {
208                     String json = builder.makeCreate(object);
209                     Entity<?> entity = Entity.entity(json,
```

```
                    MediaType.TEXT_PLAIN);
210                 Response response = this.rootTarget
211                         .path(String.format("document/%s",
                                getConfig().getDatabase().getName()))
212                         .request(MediaType.TEXT_PLAIN)
213                         .post(entity);
214                 String jsonResp = response.readEntity(String.class);
215                 try {
216                     OrientDBResponseWrapper odbrw = OM.readValue(jsonResp,
                            OrientDBResponseWrapper.class);
217                     sessionObject.put(object, odbrw.getId());
218                 } catch (IOException e) {
219                     try {
220                         LOGGER.info(OM.readValue(jsonResp,
                                OrientDBErrorWrapper.class));
221                     } catch (IOException e1) {
222                         LOGGER.error(e.getMessage(), e);
223                         throw new CheckException(e1);
224                     }
225                 }
226             }
227             if (deepness < archipelagoConfig.getDeepness()) {
228                 List<RelationWrapper> relations =
                        ArchipelagoUtils.getChilds(object);
229                 relations.stream().forEach((relationWrapper -> {
230                     Object to = relationWrapper.getTo();
231                     if (null == getId(to)) {
232                         try {
233                             persist(to, deepness + 1);
234                         } catch (CheckException e) {
235                             LOGGER.error(e.getMessage(), e);
236                         }
237                     } else {
238                         scanForLink(to, deepness + 1);
239                     }
240                     link(object, relationWrapper.getTo(),
                            relationWrapper.getName(),
                            relationWrapper.isBiDirectionnal());
241                 }));
242
243             }
244             break;
245         }
246     }
247
248     private void scanForLink(Object object, Integer deepness) {
249         if (deepness < archipelagoConfig.getDeepness()) {
250             List<RelationWrapper> relations =
                    ArchipelagoUtils.getChilds(object);
251             relations.stream().forEach((relationWrapper -> {
```

```
252                 Object to = relationWrapper.getTo();
253                 if (null == getId(to)) {
254                     try {
255                         persist(to, deepness + 1);
256                     } catch (CheckException e) {
257                         LOGGER.error(e.getMessage(), e);
258                     }
259                 } else {
260                     scanForLink(to, deepness + 1);
261                 }
262                 link(object, relationWrapper.getTo(),
                        relationWrapper.getName(),
                        relationWrapper.isBiDirectionnal());
263             }));
264         }
265     }
266
267     public void link(Object first, Object second, Object descriptor, boolean
            biDirectionnal) {
268         switch (databaseType) {
269             case NEO4J:
270                 boolean haveDescriptor = null != descriptor;
271                 try (Session s = this.driver.session()) {
272                     Integer idA = (Integer) getId(first);
273                     Integer idB = (Integer) getId(second);
274                     if (idA != null && idB != null) {
275                         String name;
276                         if (haveDescriptor) {
277                             name = descriptor.getClass().getSimpleName();
278                         } else {
279                             name = String.format("%sTo%s",
                                    first.getClass().getSimpleName(),
                                    second.getClass().getSimpleName());
280                         }
281                         if (haveDescriptor) {
282                             s.run(builder.makeRelation(idA, idB, name,
                                    descriptor.getClass()),
                                    parameters(builder.fillCreate(descriptor).toArray()));
283                             if (biDirectionnal) {
284                                 s.run(builder.makeRelation(idB, idA, name,
                                        descriptor.getClass()),
                                        parameters(builder.fillCreate(descriptor).toArray()));
285                             }
286                         } else {
287                             s.run(builder.makeRelation(idA, idB, name));
288                             if (biDirectionnal) {
289                                 s.run(builder.makeRelation(idB, idA, name));
290                             }
291                         }
292                     }
```

```
293                 }
294                 break;
295             case ORIENT_DB:
296                 LOGGER.debug("[NOT IMPLEMENTED YET]");
297                 break;
298         }
299 
300     }
301 
302     public void link(Object first, Object second, String descriptorName,
        boolean biDirectional) {
303         DescriptorWrapper dw = new DescriptorWrapper(descriptorName);
304         switch (databaseType) {
305             case NEO4J:
306                 dw.setName("".equalsIgnoreCase(descriptorName) ?
                        String.format("%sTo%s", first.getClass().getSimpleName(),
                        second.getClass().getSimpleName()) : descriptorName);
307                 try (Session s = this.driver.session()) {
308                     Integer idA = (Integer) getId(first);
309                     Integer idB = (integer) getId(second);
310                     if (idA != null && idB != null) {
311                         List<String> props = new ArrayList<>();
312                         props.add("created");
313                         s.run(builder.makeRelation(idA, idB, dw.getName(),
                                props),
                                parameters(builder.fillCreate(dw).toArray()));
314                         if (biDirectional) {
315                             s.run(builder.makeRelation(idB, idA, dw.getName(),
                                    props),
                                    parameters(builder.fillCreate(dw).toArray()));
316                         }
317                     }
318                 }
319                 break;
320             case ORIENT_DB:
321                 try {
322                     dw.setName("".equalsIgnoreCase(descriptorName) ?
                            String.format("%sTo%s",
                            first.getClass().getSimpleName(),
                            second.getClass().getSimpleName()) : descriptorName);
323                     String idA = (String) getId(first);
324                     String idB = (String) getId(second);
325                     if (idA != null && idB != null && !alreadyLinked(first,
                            second, dw)) {
326                         String query = builder.makeRelation(idA, idB, dw);
327                         LOGGER.debug(String.format("CREATE Relation from %s to
                                %s : [%s]", idA, idB, query));
328                         this.rootTarget
329                                 .path(String.format("command/%s/sql/%s",
                                        archipelagoConfig.getDatabase().getName(),
```

```
                                    URLEncoder.encode(query, "UTF-8")))
                            .request(MediaType.TEXT_PLAIN)
                            .post(Entity.entity("", MediaType.TEXT_PLAIN));
                    if (biDirectional) {
                        query = builder.makeRelation(idB, idA, dw);
                        LOGGER.debug(String.format("CREATE Relation from %s
                            to %s : [%s]", idA, idB, query));
                        this.rootTarget
                                .path(String.format("command/%s/sql/%s",
                                    archipelagoConfig.getDatabase().getName(),
                                       URLEncoder.encode(query, "UTF-8")))
                                .request(MediaType.TEXT_PLAIN)
                                .post(Entity.entity("",
                                    MediaType.TEXT_PLAIN));
                    }
                }
            } catch (UnsupportedEncodingException e) {
                LOGGER.error(e.getMessage(), e);
            }
            break;
        }
    }

    private boolean alreadyLinked(Object first, Object second,
        DescriptorWrapper dw) {
        switch (databaseType) {
            case NEO4J:
                break;
            case ORIENT_DB:
                try {
                    String firstJson = this.rootTarget
                            .path(String.format("command/%s/sql/%s",
                                archipelagoConfig.getDatabase().getName(),
                                   URLEncoder.encode(builder.makeMatch(first),
                                       "UTF-8")))
                            .request(MediaType.APPLICATION_JSON)
                            .get(String.class);
                    String idSecond = (String) getId(second);
                    OrientDBResultWrapper firstResult = null;

                    firstResult = OM.readValue(firstJson,
                        OrientDBResultWrapper.class);

                    for (Map<String, Object> map : firstResult.getResult()) {
                        List<String> links = (List<String>)
                            map.get(String.format("out_%s", dw.getName()));
                        if (null != links) {
                            for (String linked : links) {
                                String linkJson = this.rootTarget
                                        .path(String.format("command/%s/sql/%s",
```

```
                                        archipelagoConfig.getDatabase().getName(),
372                                        URLEncoder.encode(String.format("SELECT
                                               FROM %s", linked), "UTF-8")))
373                                    .request(MediaType.APPLICATION_JSON)
374                                    .get(String.class);
375                            OrientDBResultWrapper odbrw =
                                   OM.readValue(linkJson,
                                   OrientDBResultWrapper.class);
376                            for (Map<String, Object> props :
                                   odbrw.getResult()) {
377                                if (idSecond.equalsIgnoreCase((String)
                                       props.get("in"))) {
378                                    LOGGER.debug(String.format("%s and %s are
                                           already linked", map.get("@rid"),
                                           idSecond));
379                                    return true;
380                                }
381                            }
382                        }
383                    }
384                }
385            } catch (Exception e) {
386                e.printStackTrace();
387            }
388            break;
389        }
390        return false;
391    }
392
393    /**
394     * Persits an object in the database
395     *
396     * @param object the object to persist
397     * @return the internal Id of the object
398     */
399    public Object getId(Object object) {
400        if (sessionObject.containsKey(object)) return
               sessionObject.get(object);
401        Object id = null;
402        Object[] params = builder.fillCreate(object).toArray();
403        switch (databaseType) {
404            case NEO4J:
405
406                try (Session s = this.driver.session()) {
407                    String stmt = builder.makeMatch(object, false);
408                    StatementResult result = s.run(stmt, parameters(params));
409                    while (result.hasNext()) {
410                        Record record = result.next();
411                        id = record.get("InternalId").asInt();
412                    }
```

```
413                 } catch (Exception e) {
414                     LOGGER.error(String.format("Params [%s]",
                            StringUtils.join(params, ',')));
415                     LOGGER.error(e.getMessage(), e);
416                     throw e;
417                 }
418                 break;
419             case ORIENT_DB:
420                 try {
421                     String json = this.rootTarget
422                             .path(String.format("command/%s/sql/%s",
                                    archipelagoConfig.getDatabase().getName(),
423                                        URLEncoder.encode(builder.makeMatch(object),
                                            "UTF-8")))
424                             .request(MediaType.APPLICATION_JSON)
425                             .get(String.class);
426                     OrientDBResultWrapper resultWrapper = OM.readValue(json,
                            OrientDBResultWrapper.class);
427                     for (Map<String, Object> map : resultWrapper.getResult()) {
428                         id = map.get("@rid");
429                     }
430                 } catch (Exception e) {
431                     LOGGER.error(String.format("Params [%s]",
                            StringUtils.join(params, ',')));
432                     LOGGER.error(e.getMessage(), e);
433                 }
434                 break;
435             default:
436         }
437         if (null != id) sessionObject.put(object, id);
438         return id;
439     }
440
441     @Override
442     public void close() throws Exception {
443         driver.close();
444     }
445
446     public List<Object> execute(ArchipelagoQuery aq) throws CheckException,
            IOException {
447         List<Object> nodes = new LinkedList<>();
448         LOGGER.debug(String.format("Query : %s", aq.getQuery()));
449         final Map<Long, Object> startNodes = new HashMap<>();
450         switch (databaseType) {
451             case NEO4J:
452                 try (Session s = this.driver.session()) {
453                     if (aq.isRelation()) {
454                         link(aq.getFrom(), aq.getTo(), aq.getDescriptor(),
                                aq.isBiDirectionnal());
455                     } else {
```

```
456                     StatementResult result = s.run(aq.getQuery());
457                     while (result.hasNext()) {
458                         Record record = result.next();
459                         Path p = record.get("p").asPath();
460                         p.forEach(segment -> {
461                             try {
462                                 Object startNode = null;
463                                 Node start = segment.start();
464                                 if (!startNodes.containsKey(start.id())) {
465                                     startNode =
                                             aq.getTarget().getConstructor().newInstance();
466                                     ArchipelagoUtils.feedObject(startNode,
                                             start.asMap());
467                                     startNodes.put(start.id(), startNode);
468
469                                 } else {
470                                     startNode = startNodes.get(start.id());
471                                 }
472                                 Relationship rel = segment.relationship();
473                                 Field field =
                                      ArchipelagoUtils.getFieldFromBridgeName(startNode.g
                                      rel.type());
474                                 //Get the label as the ClassName
475                                 Set<Class<?>> supers =
                                      this.reflections.getSubTypesOf(ArchipelagoUtils.get
476                                 Class<?> endClass = null;
477                                 if (supers.size() > 0) {
478                                     endClass = ArchipelagoUtils
479                                             .getClassOf(supers,
                                                 segment.end().labels().iterator().next()
480                                 } else {
481                                     endClass =
                                             ArchipelagoUtils.getClassOf(field);
482                                 }
483                                 Object endNode = endClass
484                                         .getConstructor()
485                                         .newInstance();
486                                 ArchipelagoUtils.feedObject(endNode,
                                      segment.end().asMap());
487                                 if
                                      (Collection.class.isAssignableFrom(field.getType())
                                      {
488                                     ((Collection)
                                             ArchipelagoUtils.get(aq.getTarget(),
                                             field, startNode)).add(endNode);
489                                 } else {
490                                     String methodName =
                                             String.format("set%s%s", ("" +
                                             field.getName().charAt(0)).toUpperCase(),
                                             field.getName().substring(1,
```

```
                                    field.getName().length()));
491                                 //Must have exactly one match
492                                 Method method =
                                        ReflectionUtils.getAllMethods(aq.getTarget(),
493                                         withModifier(Modifier.PUBLIC),
                                                withPrefix(String.format(methodName)))
494                                         .stream()
495                                         .findFirst()
496                                         .get();
497                                 method.invoke(startNode, endNode);
498                             }
499                         } catch (Exception e) {
500                             LOGGER.error(e.getMessage(), e);
501                         }
502                     });
503                 }
504             }
505         } catch (Exception e) {
506             LOGGER.error(e.getMessage(), e);
507             throw new CheckException(e.getMessage());
508         }
509         nodes.addAll(startNodes.values());
510         break;
511     case ORIENT_DB:
512         try {
513             String json;
514             if (aq.isRelation()) {
515                 json = this.rootTarget
516                         .path(String.format("command/%s/sql/%s",
                                archipelagoConfig.getDatabase().getName(),
517                                 URLEncoder.encode(aq.getQuery(),
                                        "UTF-8")))
518                         .request(MediaType.TEXT_PLAIN)
519                         .post(Entity.entity("", MediaType.TEXT_PLAIN),
                                String.class);
520             } else {
521                 json = this.rootTarget
522                         .path(String.format("command/%s/sql/%s",
                                archipelagoConfig.getDatabase().getName(),
523                                 URLEncoder.encode(aq.getQuery(),
                                        "UTF-8")))
524                         .request(MediaType.APPLICATION_JSON)
525                         .get(String.class);
526             }
527
528             OrientDBResultWrapper resultWrapper = OM.readValue(json,
                    OrientDBResultWrapper.class);
529             //TODO Too complicatted O(n^5) omg
530             for (Map<String, Object> map : resultWrapper.getResult()) {
531                 Object node =
```

```
                    aq.getTarget().getConstructor().newInstance();
532                 ArchipelagoUtils.feedObject(node, map);
533                 nodes.add(node);
534                 for (Map.Entry<String, Object> prop : map.entrySet()) {
535                     if (StringUtils.startsWith(prop.getKey(), "out_")) {
536                         String name = prop.getKey().substring(4);
537                         Field field =
                                ArchipelagoUtils.getFieldFromBridgeName(aq.getTarget(),
                                name);
538                         //Get the label as the ClassName
539                         Set<Class<?>> supers =
                                this.reflections.getSubTypesOf(ArchipelagoUtils.getClas
540                         for (String relations : (List<String>)
                                prop.getValue()) {
541                             String relJson = this.rootTarget
542                                     .path(String.format("command/%s/sql/%s",
                                            archipelagoConfig.getDatabase().getName(),
543                                            URLEncoder.encode(String.format("SELECT
                                                FROM %s", relations),
                                                "UTF-8")))
544                                     .request(MediaType.APPLICATION_JSON)
545                                     .get(String.class);
546                             OrientDBResultWrapper odbrw =
                                    OM.readValue(relJson,
                                    OrientDBResultWrapper.class);
547                             for (Map<String, Object> props :
                                    odbrw.getResult()) {
548                                 String linked = (String) props.get("in");
549                                 String linkedJson = this.rootTarget
550                                         .path(String.format("command/%s/sql/%s",
                                                archipelagoConfig.getDatabase().getName()
551                                                URLEncoder.encode(String.format("SELE
                                                    FROM %s", linked),
                                                    "UTF-8")))
552                                         .request(MediaType.APPLICATION_JSON)
553                                         .get(String.class);
554                                 OrientDBResultWrapper link =
                                        OM.readValue(linkedJson,
                                        OrientDBResultWrapper.class);
555                                 for (Map<String, Object> linkedProps :
                                        link.getResult()) {
556                                     Class<?> endClass = null;
557                                     if (supers.size() > 0) {
558                                         endClass = ArchipelagoUtils
559                                                 .getClassOf(supers,
                                                        (String)
                                                        linkedProps.get("@class"));
560                                     } else {
561
562                                         // TODO reTest
```

```
563                                         endClass =
                                                ArchipelagoUtils.getClassOf(field);
564                                     }
565                                     Object endNode = endClass
566                                             .getConstructor()
567                                             .newInstance();
568                                     ArchipelagoUtils.feedObject(endNode,
                                            linkedProps);
569                                     if
                                            (Collection.class.isAssignableFrom(field.get
                                            {
570                                         ((Collection)
                                                ArchipelagoUtils.get(aq.getTarget(),
                                                field, node)).add(endNode);
571                                     } else {
572                                         String methodName =
                                                String.format("set%s%s", (""
                                                +
                                                field.getName().charAt(0)).toUpperCase()
                                                field.getName().substring(1,
                                                field.getName().length()));
573                                         //Must have exactly one match
574                                         Method method =
                                                ReflectionUtils.getAllMethods(aq.getTarg
575                                                    withModifier(Modifier.PUBLIC),
                                                        withPrefix(String.format(methodNa
576                                                    .stream()
577                                                    .findFirst()
578                                                    .get();
579                                         method.invoke(node, endNode);
580                                     }
581                                 }
582                             }
583                         }
584                     }
585                 }
586             }
587         } catch (Exception e) {
588             LOGGER.error(e.getMessage(), e);
589             throw new CheckException(e.getMessage());
590         }
591         break;
592     }
593     return nodes;
594 }
595 }
```

ArchipelagoUtils.java

```
1 package org.archipelago.core.util;
2
```

```
3  import org.apache.commons.io.FilenameUtils;
4  import org.apache.commons.lang3.AnnotationUtils;
5  import org.apache.logging.log4j.LogManager;
6  import org.apache.logging.log4j.Logger;
7  import org.archipelago.core.annotations.Bridge;
8  import org.archipelago.core.domain.RelationWrapper;
9
10 import javax.tools.JavaCompiler;
11 import javax.tools.ToolProvider;
12 import java.io.File;
13 import java.io.IOException;
14 import java.lang.annotation.Annotation;
15 import java.lang.reflect.*;
16 import java.net.MalformedURLException;
17 import java.net.URL;
18 import java.nio.file.Files;
19 import java.nio.file.Path;
20 import java.nio.file.Paths;
21 import java.text.SimpleDateFormat;
22 import java.time.LocalDate;
23 import java.time.LocalDateTime;
24 import java.time.LocalTime;
25 import java.time.format.DateTimeFormatter;
26 import java.util.*;
27 import java.util.regex.Matcher;
28 import java.util.regex.Pattern;
29
30 import static org.archipelago.core.framework.Archipelago.ARCHIPELAGO_ID;
31
32 /**
33  * @author Gilles Bodart
34  */
35 public class ArchipelagoUtils {
36
37     public static final String JAVA_EXTENSION = "java";
38     public static final String CLASS_EXTENSION = "class";
39     private static final Pattern pattern = Pattern.compile("\\w+To(\\w+)");
40     private final static Logger LOGGER =
          LogManager.getLogger(ArchipelagoUtils.class);
41
42     public static boolean doesContainsAnnotation(List<Annotation>
          annotations, Class<? extends Annotation>... classes) {
43         for (Annotation annotation : annotations) {
44             for (Class<?> clazz : classes) {
45                 if (clazz.equals(annotation.annotationType())) {
46                     return true;
47                 }
48             }
49         }
50         return false;
```

```
51      }
52
53      public static boolean doesContainsAnnotation(Annotation[] annotations,
            Class<? extends Annotation>... classes) {
54          return doesContainsAnnotation(Arrays.asList(annotations), classes);
55      }
56
57      public static void compile(final List<File> javaClasses) {
58          final JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
59          if (null != javaClasses) {
60              for (File javaClass : javaClasses) {
61                  compiler.run(null, null, null, javaClass.getPath());
62              }
63          }
64      }
65
66      public static void clean(final Path domainFolder) throws IOException {
67          final List<File> javaCompiledClasses = analyse(domainFolder,
                CLASS_EXTENSION);
68          if (null != domainFolder) {
69              for (File compiledClass : javaCompiledClasses) {
70
71                  Files.deleteIfExists(Paths.get(compiledClass.toURI()));
72              }
73          }
74      }
75
76      public static URL[] retriveURL(final Path domainFolder, final List<File>
            javaClasses) throws MalformedURLException {
77          final URL[] urls = new URL[javaClasses.size()];
78          for (int i = 0; i < javaClasses.size(); i++) {
79              urls[i] = javaClasses.get(i).toURI().toURL();
80          }
81          return urls;
82      }
83
84      public static List<File> analyse(final Path folder, final String
            extension) {
85          assert (folder.toFile().isDirectory());
86          final List<File> files = new LinkedList<>();
87
88          for (File element : folder.toFile().listFiles()) {
89              if
                    (FilenameUtils.getExtension(element.getName()).equalsIgnoreCase(extension
                    {
90                  files.add(element);
91              }
92          }
93          return files;
94      }
```

```
95
96      public static void scan(final Class<?> clazz) {
97          for (Annotation x : clazz.getAnnotations()) {
98              LOGGER.debug(String.format("Annotation : %s",
                    AnnotationUtils.toString(x)));
99          }
100         for (Field x : clazz.getDeclaredFields()) {
101             LOGGER.debug(String.format("Field : %s", x.getName()));
102         }
103         for (Method x : clazz.getDeclaredMethods()) {
104             LOGGER.debug(String.format("Method : %s", x.getName()));
105         }
106         for (Constructor<?> x : clazz.getDeclaredConstructors()) {
107             LOGGER.debug(String.format("Constructor : %s", x.getName()));
108         }
109     }
110
111     public static Set<Field> getAllFields(Class<?> clazz) {
112         Set<Field> fields = new HashSet<>();
113
114         fields.addAll(Arrays.asList(clazz.getDeclaredFields()));
115
116         Class<?> superClazz = clazz.getSuperclass();
117         if (superClazz != null) {
118             fields.addAll(getAllFields(superClazz));
119         }
120
121         return fields;
122     }
123
124
125     public static void feedObject(Object o, Map<String, Object> properties) {
126         for (Map.Entry<String, Object> entry : properties.entrySet()) {
127             String fieldName = entry.getKey();
128             if (!ARCHIPELAGO_ID.equalsIgnoreCase(fieldName) && null !=
                        entry.getValue() && !"@".equalsIgnoreCase("" +
                        fieldName.charAt(0))) {
129                 Method setter = null;
130                 String methodName = String.format("set%s%s", ("" +
                            fieldName.charAt(0)).toUpperCase(), fieldName.substring(1,
                            fieldName.length()));
131                 try {
132                     if (Integer.class.equals(entry.getValue().getClass())) {
133                         try {
134                             setter = o.getClass().getMethod(methodName,
                                        Integer.class);
135                             setter.invoke(o, Integer.valueOf("" +
                                        entry.getValue()));
136                         } catch (NoSuchMethodException e) {
137                             setter = o.getClass().getMethod(methodName,
```

```
                            Long.class);
                    setter.invoke(o, Long.valueOf("" +
                            entry.getValue()));
                }
            } else {
                setter = o.getClass().getMethod(methodName,
                    entry.getValue().getClass());
                setter.invoke(o, entry.getValue());
            }
        } catch (Exception e) {
            LOGGER.debug(String.format("Method not found %s with param
                %s", methodName, entry.getValue().getClass()));
        }
    }
}

public static Object formatQueryValue(Object o) {
    return formatQueryValue(o, false);
}

public static Object formatQueryValue(Object o, boolean stringDate) {
    return formatQueryValue(o, stringDate, false);
}
public static Object formatQueryValue(Object o, boolean stringDate,
    boolean appendString) {
    Object formated = o;
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-mm-dd");
    if (formated instanceof String && appendString) {
        formated = String.format("\"%s\"", formated);
    } else {
        String elm = null;
        switch (o.getClass().getSimpleName().toLowerCase()) {
            case "date":
                elm = sdf.format((Date) o);
                formated = stringDate ? String.format("\"%s\"", elm) : elm;
                break;
            case "localdate":
                elm = ((LocalDate)
                    o).format(DateTimeFormatter.ISO_LOCAL_DATE);
                formated = stringDate ? String.format("\"%s\"", elm) : elm;
                break;
            case "localtime":
                elm = ((LocalTime)
                    o).format(DateTimeFormatter.ISO_LOCAL_DATE);
                formated = stringDate ? String.format("\"%s\"", elm) : elm;
                break;
            case "localdatetime":
                elm = ((LocalDateTime)
                    o).format(DateTimeFormatter.ISO_LOCAL_DATE);
```

```
                formated = stringDate ? String.format("\"%s\"", elm) : elm;
                break;
            default: {
                formated = o;
            }
        }
    }
    return formated;
}


public static List<RelationWrapper> getChilds(Object object) {
    List<RelationWrapper> relations = new ArrayList<>();
    Class<?> clazz = object.getClass();
    for (Field field : getAllFields(clazz)) {
        if (field.isAnnotationPresent(Bridge.class)) {
            try {
                Method getter = null;
                if (field.getType().equals(boolean.class)) {
                    getter = clazz.getMethod(String.format("is%s%s", ("" +
                        field.getName().charAt(0)).toUpperCase(),
                        field.getName().substring(1, field
                            .getName().length())));
                } else {
                    getter = clazz.getMethod(String.format("get%s%s", ("" +
                        field.getName().charAt(0)).toUpperCase(),
                        field.getName().substring(1, field
                            .getName().length())));
                }
                Object prop = getter.invoke(object);
                List<Object> props = new ArrayList<>();
                if (null != prop) {
                    if (prop instanceof Collection) {
                        Iterator i = ((Collection) prop).iterator();
                        while (i.hasNext()) {
                            Object child = i.next();
                            props.add(child);
                        }
                    } else {
                        props.add(prop);
                    }
                }
                props.stream().forEach(p -> {
                    RelationWrapper rw = new RelationWrapper();
                    rw.setName(field.getAnnotation(Bridge.class).descriptor());
                    rw.setTo(p);
                    rw.setBiDirectionnal(field.getAnnotation(Bridge.class).biDirection
                    relations.add(rw);
                });
            } catch (NoSuchMethodException | SecurityException |
```

```
                    IllegalAccessException | IllegalArgumentException |
                    InvocationTargetException e) {
226                 LOGGER.debug(String.format("No usual getter for %s found",
                        field.getName()), e);
227             }
228         }
229     }
230     return relations;
231 }
232
233 public static Object get(Class<?> clazz, Field field, Object object) {
234     Method getter = null;
235     try {
236         if (field.getType().equals(boolean.class) ||
                field.getType().equals(Boolean.class)) {
237             getter = clazz.getMethod(String.format("is%s%s", ("" +
                    field.getName().charAt(0)).toUpperCase(),
                    field.getName().substring(1, field
238                         .getName().length())));
239         } else {
240             getter = clazz.getMethod(String.format("get%s%s", ("" +
                    field.getName().charAt(0)).toUpperCase(),
                    field.getName().substring(1, field
241                         .getName().length())));
242         }
243         return getter.invoke(object);
244     } catch (Exception e) {
245         LOGGER.debug(String.format("No usual getter for %s found",
                field.getName()), e);
246         LOGGER.error(e.getMessage(), e);
247     }
248     return null;
249 }
250
251
252 public static Class getClassOf(Field field) {
253     Class clazz;
254     if (Collection.class.isAssignableFrom(field.getType())) {
255         ParameterizedType genericType = (ParameterizedType)
                field.getGenericType();
256         if (genericType.getActualTypeArguments()[0] instanceof
                WildcardType) {
257             clazz = (Class<?>) ((WildcardType)
                    genericType.getActualTypeArguments()[0]).getUpperBounds()[0];
258         } else {
259             clazz = (Class<?>) genericType.getActualTypeArguments()[0];
260         }
261     } else {
262         clazz = field.getType();
263     }
```

```
264         return clazz;
265     }
266 
267     public static Class<?> getClassOf(Set<Class<?>> classes, String
            className) {
268         for (Class clazz : classes) {
269             if (clazz.getSimpleName().equalsIgnoreCase(className)) {
270                 return clazz;
271             }
272         }
273         return null;
274     }
275 
276     public static Field getFieldFromBridgeName(Class<?> clazz, String type) {
277         Field field = null;
278         Matcher m = pattern.matcher(type);
279         Set<Field> fields = getAllFields(clazz);
280 
281         for (Field f : fields) {
282             if (f.isAnnotationPresent(Bridge.class)
283                     &&
                            type.equalsIgnoreCase(f.getAnnotation(Bridge.class).descriptor()))
                            {
284                 return f;
285             }
286         }
287         return field;
288     }
289 }
```

StringTemplateFactory.java

```
1  package org.archipelago.core.util;
2 
3  import org.stringtemplate.v4.*;
4  import org.stringtemplate.v4.misc.ErrorBuffer;
5 
6  import java.util.Date;
7 
8  /**
9   *
10  * @author Gilles Bodart
11  *
12  */
13 public class StringTemplateFactory {
14 
15     public static STGroup buildSTGroup(final String ressourcePath) {
16 
17         final STGroup group = new STGroupFile(ressourcePath);
18         group.registerRenderer(String.class, new StringRenderer());
19         group.registerRenderer(Integer.class, new NumberRenderer());
```

```
20          group.registerRenderer(Date.class, new DateRenderer());
21          final ErrorBuffer errors = new ErrorBuffer();
22          group.setListener(errors);
23          group.load();
24          return group;
25      }
26  }
```

Bibliographie

- https://neo4j.com/ consulté à de nombreuses reprises (Neo Technology, Inc)
- https://orientdb.com/ consulté à de nombreuses reprises (OrientDB LTD)
- https://snap.stanford.edu/data/
- https://networkx.github.io/
- http://igraph.org/redirect.html
- https://snap.stanford.edu/data/egonets-Facebook.html
- http://konect.uni-koblenz.de/
- https://icon.colorado.edu/#!/networks
- https://neonx.readthedocs.io/en/latest/
- J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.
- J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29–123, 2009.
- https://www.infoq.com/fr/articles/graph-nosql-neo4j
- http://www.silicon.fr/base-donnees-nosql-impose-sgbdr-93305.html
- https://prezi.com/4flswlgipwbo/nosql-not-only-sql/

LIVRE http://www.eyrolles.com/Chapitres/9782212141559/9782212141559.pdf

- https://db-engines.com
- https://www.udemy.com/orientdb-getting-started
- https://fr.wikipedia.org/wiki/Representational_state_transfer