République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: Génie logiciel (G.L)

Thème

Intégration des filtres spatiaux au sein du triplestore RDF_QDAG

Réalisé par :

- Mohammed El Bachir BOUCHENAKI

Présenté le [26/06/2023] devant le jury composé de :

- M. Amine BELABED (Président)
- M. Houcine MATALLAH (Encadrant)
- M. Nadir GUERMOUDI (Co-encadrant)
- Mme. Amina BENOSMAN (Examinatrice)

Année universitaire : 2022-2023

Remerciements

En préambule à ce mémoire, je remercie Dieu le tout puissant et miséricordieux qui m'a donné la force et la patience d'accomplir ce travail.

Je tiens à remercier sincèrement mes encadrants M. Houcine MATALLAH et M. Nadir GUERMOUDI d'avoir enrichi mes connaissances et de m'avoir guidé durant ce travail. Je les remercie aussi pour tous leurs précieux conseils, pour leur écoutes actives.

L'expression de mes remerciements les plus sincères va aux membres de notre jury M. Amine BELABED et Mme. Amina BENOSMAN pour avoir accepté de lire ce mémoire et d'évaluer notre travail.

Merci à tous les enseignants du département informatique qui ont fourni des efforts pour nous donner leurs savoirs durant tout notre cursus universitaire.

À toutes les personnes qui m'ont aidé durant la réalisation de ce travail, veuillez trouver dans ces lignes l'expression de ma reconnaissance et ma profonde gratitude.

Mohammed El Bachir BOUCHENAKI

Dédicaces

Je dédie ce modeste travail en signe de respect, reconnaissance et de remerciement aux personnes les plus chères à mon cœur :

À mes très chers parents qui m'ont soutenu et qui ont su m'aider à atteindre mes objectifs et pour tous les sacrifices qu'ils ont fait pour moi. Que ce travail soit le témoignage de ma plus profonde reconnaissance envers eux.

À mes très chères sœurs Meryem et Manel ainsi que mon très cher frère Chiheb Eddine qui ont partagé avec moi tous les moments d'émotion lors de la réalisation de ce travail. Ils m'ont chaleureusement supporté et encouragé tout au long de mon parcours.

À tous les gens que j'ai connu dans ma vie et qui m'ont aidé d'une quelconque façon.

Mohammed El Bachir BOUCHENAKI

Table des matières

1 Introduction **3**

1.1 Contexte du projet 3

1.2 Problématique 4

1.3 Objectifs 5

1.4 Intégration de l'équipe RDF_QDAG 5

2 Analyse de l'existant : RDF_QDAG **6**

2.1 Introduction 6

2.2 Représentation des données spatiales avec RDF_QDAG 6

2.3 L'utilisation du WKT 7

2.4 La syntaxe d'une requête SPARQL spatiale 8

2.5 Le parsing des requetes spatiales 10

2.6 La bibliothèque JTS 11

2.7 Le bulk runner 12

3 Conception **13**

3.1 Introduction 13

3.2 La création d'un filtre 13

3.3 Le patron de conception 17

3.4 L'intégration continue 19

4 Réalisation **23**

4.1 Introduction 23

4.2 Implémentation des filtres 23

4.2.1 Le filtre Within 23

4.2.2 Le filtre Touches 24

4.2.3 Le filtre Overlaps 24
4.2.4 Le filtre Contains 24
4.2.5 Le filtre Equals 24
4.3 Exemple de l'implémentation du filtre 'Overlaps' 25
4.4 Les tests unitaires des filtres 29
4.4.1 Introduction 29
4.4.2 Exemple détaillé 29
4.5 Le fichier log du bulk runner 32

5 Gestion de projet **35**
5.1 Introduction 35
5.2 Outils collaboratifs 35
5.2.1 Overleaf 35
5.2.2 GitHub 36
5.2.3 Docker 36
5.2.4 Microsoft Teams 37
5.3 Modèle de développement et planning 37
5.4 Conclusion 40

6 Conclusion et perspectives **41**
6.1 Conclusion 41
6.2 Perspectives 42

Chapitre 1

Introduction

Notre projet de fin d'études s'inscrit dans le cadre d'une collaboration entre le laboratoire LRIT de l'université de Tlemcen (Algérie) et le laboratoire LIAS de l'université de Poitiers (France). Nous avons rejoint l'équipe de recherche qui développe le Triplestore RDF_QDAG (http ://qdag.projets.univ-poitiers.fr/) pour intégrer de nouveaux opérateurs de calcul spatial qui ne sont pas supportés pour le moment par RDF_QDAG.

1.1 Contexte du projet

L'interconnexion massive des données, des informations et des connaissances a permis de créer ce qu'on appelle aujourd'hui les graphes de connaissances [8]. Rapidement, ce concept a été étendu à d'autres domaines (e.g., média [1] , industries automobile [19] et pharmaceutique [2] , biologie [14]). Les graphes de connaissances permettent aujourd'hui d'apporter une certaine sémantique pour donner un contexte et des relations aux données, tout en fournissant un cadre standard pour l'intégration, l'unification, l'analyse et le partage des données (https ://hbr.org/sponsored/2019/04/how-third-party-information-can-enhance-data-analytics). Dans ce contexte, le Resource Description Framework (RDF) et SPARQL se sont distingués respectivement pour représenter les données et les interroger.

1. https://www.slideshare.net/ConnectedDataLondon/ten-years-of-linked-data-at-the-bbc
2. https://www.ontotext.com/knowledgehub/case-studies/pharma-company-uses-ontotexts-smartersearch-across-siloed-data

Une donnée spatiale est une information qui est liée à une position géographique spécifique. Elle est généralement représentée sous forme de coordonnées géographiques. Les données spatiales sont utilisées dans divers domaines pour de nombreuses applications. Elles sont précieuses dans la planification urbaine [6], permettant la cartographie et la modélisation des infrastructures urbaines. Dans le domaine de l'environnement et de la gestion des ressources naturelles[9], ces données sont indispensables pour surveiller les changements climatiques, analyser la déforestation, suivre les migrations animales, et prévoir les risques naturels tels que les inondations et les feux de forêt.

1.2 Problématique

La nécessité de gérer et interroger efficacement les données RDF a conduit au développement de nouveaux systèmes, appelés "Triplestores", qui sont conçus spécialement pour traiter ce format de données. Malgré le nombre important de Triplestores proposés dans la littérature, il est difficile de trouver une solution qui offre des temps de réponse acceptables pour des requêtes SPARQL lorsqu'il s'agit de gérer plusieurs milliards de triplets RDF [10]. En fait, ces systèmes s'appuient sur des techniques de stockage, d'évaluation et d'optimisation de requêtes qui ne sont pas en adéquation avec la nature des données RDF à cause de l'absence d'un schéma explicite de données. En outre, les Triplestores existants peinent aussi à offrir des garanties sur les performances, même quand il s'agit de traiter des requêtes avec des opérateurs standards (e.g., BGP, WildCard et agrégation). Le seul système permettant de garantir un certain compromis entre passage à l'échelle et performances est, à notre connaissance, l'approche centralisée RDF_QDAG [10] [3], basée sur l'exploration de graphe et la fragmentation.

3. https ://qdag.projets.univ-poitiers.fr/qdag/

1.3 Objectifs

L'objectif central de notre projet de fin d'études est d'intégrer avec succès des filtres spatiaux au sein du triplestore RDF_QDAG. Pour atteindre cet objectif, nous examinerons attentivement les algorithmes existants et chercherons la stratégie la mieux adaptée à RDF_QDAG, garantissant ainsi des performances optimales.

1.4 Intégration de l'équipe RDF_QDAG

L'équipe du projet RDF_QDAG regroupe plusieurs chercheurs issus d'universités françaises, algériennes et tunisiennes [4]. Une vingtaine d'étudiants en Licence et en Master ont travaillé sur ce projet afin de valider leurs diplômes.

Dans le cadre de notre projet de fin d'études, nous avons collaboré étroitement avec deux doctorants, M. Nadir GUERMOUDI et M. Houssameddine YOUSFI , qui se sont concentrés sur la partie spatiale de RDF_QDAG. Notre proposition les soutient dans la validation de leurs travaux de recherche. De plus, nous avons bénéficié de la collaboration du Dr. Amin MESMOUDI, qui a apporté son expertise dans le domaine.

4. https://qdag.projets.univ-poitiers.fr/ qdag/contact

Chapitre 2

Analyse de l'existant : RDF_QDAG

2.1 Introduction

Dans cette section, nous décrivons les concepts et techniques clés utilisés dans la partie spatiale de RDF_QDAG, et nous donnons quelques détails sur les filtres spatiaux.

2.2 Représentation des données spatiales avec RDF_QDAG

RDF_QDAG offre des fonctionnalités avancées pour la représentation des données spatiales. Les données spatiales sont des informations géographiques qui décrivent la position, la forme ou les caractéristiques d'objets dans l'espace. Pour représenter ces données dans un RDF_QDAG, on utilise des ontologies géospatiales qui définissent des concepts tels que les coordonnées géographiques [2], les limites géographiques, les entités géographiques, etc.

Dans RDF_QDAG, les entités géographiques sont généralement représentées comme des ressources avec des propriétés spécifiques. Par exemple, un lieu peut être représenté en tant que nœud avec des propriétés telles que son nom, ses coordonnées géographiques, sa catégorie (par exemple, ville, pays, monument, etc.) et d'autres attributs pertinents.

L'utilisation de RDF_QDAG permet de créer un réseau d'informations spatiales interconnectées et de réaliser des requêtes complexes qui exploitent les relations spatiales entre les entités. L'utilisation de RDF_QDAG pour la représentation des données spatiales facilite l'interrogation et l'analyse des données géospatiales, offrant ainsi de nouvelles perspectives dans des domaines tels que la géographie, l'aménagement du territoire et la navigation.

2.3 L'utilisation du WKT

Le WKT (Well-Known Text) est un format de représentation textuelle standard utilisé pour décrire les géométries géospatiales [15]. Il fournit une syntaxe simple et lisible par les humains pour représenter des entités géométriques. Voici la représentation de ces entités avec un exemple de chaque entité :

- Point (Point) : POINT (30 10)
- Ligne (LineString) : LINESTRING (30 10, 10 30, 40 40)
- Polygone (Polygon) : POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
- Multipoint (MultiPoint) : MULTIPOINT ((10 40), (40 30), (20 20), (30 10))
- Multiligne (MultiLineString) : MULTILINESTRING ((10 10, 20 20, 10 40), (40 40, 30 30, 40 20, 30 10))
- Multipolygone (MultiPolygon) : MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))
- Collection géométrique (GeometryCollection) : GEOMETRYCOLLECTION (POINT (10 10), LINESTRING (0 0, 20 20, 25 25), POLYGON ((0 0, 10 10, 10 0, 0 0)))

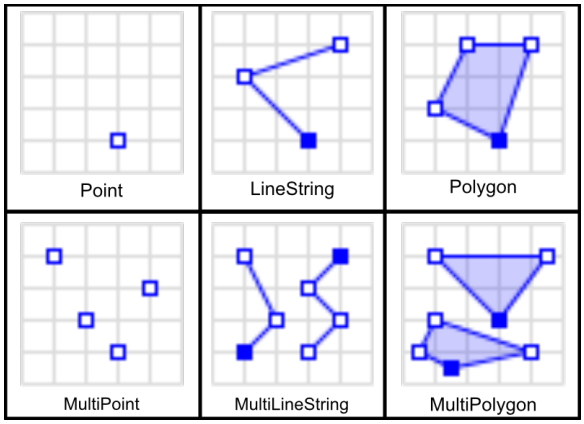


FIGURE 2.1 – Rprésentation des entités du WKT

Les coordonnées sont fournies sous forme de paires de valeurs (x, y), où x représente la coordonnée horizontale (longitude) et y représente la coordonnée verticale (latitude) dans un système de coordonnées donné.

On remarque aussi que dans la représentation WKT d'un polygone, la dernière coordonnée doit être identique à la première coordonnée pour fermer la forme géométrique. Cela indique que le polygone est une forme fermée, où le dernier point est connecté au premier point.

2.4 La syntaxe d'une requête SPARQL spatiale

Les requêtes SPARQL spatiales sont des requêtes qui utilisent le langage de requête SPARQL (Simple Protocol and RDF Query Language) pour interroger des données géospatiales [13] stockées sous forme de graphes RDF (Resource Description Framework). Ces requêtes permettent de rechercher et d'analyser des informations géospatiales en combinant des critères spatiaux avec des conditions sémantiques [12] .

Le langage SPARQL comprend des extensions spécifiques pour les requêtes spatiales, notamment des opérations de filtrage [5] et de comparaison basées sur des relations spatiales telles que l'intersection, la proximité, l'inclusion, etc. Ces opérations permettent de définir des conditions spatiales dans les requêtes pour trouver des entités géospatiales qui répondent à des critères spécifiques.
Voici un exemple d'une requête SPARQL spatial :

```
SELECT ?g WHERE {
  ?o <http://linkedgeodata.org/ontology/boat> ?b .
  ?o <http://geovocab.org/geometry#geometry> ?p .
  ?p <http://www.opengis.net/ont/geosparql#asWKT> ?g .
}
FILTER ?g INTERSECTS "POLYGON((-100 20, -80 20, -80 40, -100 40, -100 20))";
```

FIGURE 2.2 – Requête SPARQL spatial

La syntaxe de cette requête est utilisée pour sélectionner les valeurs de la variable ' ?g' à partir d'un graphe RDF qui satisfait certaines conditions. Voici une explication de la syntaxe de cette requête :

- SELECT ?g' : Cela indique que nous souhaitons sélectionner la valeur de la variable ?g dans les résultats de la requête. Cette variable représentera les géométries WKT des résultats.
- 'WHERE ... ' : Cette clause spécifie les motifs de triplets à rechercher. Dans ce cas, nous avons trois triplets imbriqués.
- ' ?o <http ://linkedgeodata.org/ontology/boat> ?b' : Ce triplet recherche des triplets où une ressource ' ?o' est liée à la propriété '<http ://linkedgeodata.org/ontology/boat>' avec une valeur ' ?b'. Cela permet de filtrer les ressources qui sont liées à la classe ou à l'objet "boat" dans l'ontologie Linkedgeodata.
- ' ?o <http ://geovocab.org/geometry#geometry> ?p' : Ce triplet recherche des triplets où la même ressource ' ?o' est liée à la propriété '<http ://geovocab.org/geometry#geometry>' avec une valeur ' ?p'. Cela permet de filtrer les ressources qui ont une géométrie associée.
- ' ?p <http ://www.opengis.net/ont/geosparql#asWKT> ?g' : Ce triplet recherche des triplets où la variable ' ?p' est liée à la propriété '<http ://www.opengis.net/ont/geosparql#asWKT>' avec une valeur ' ?g'. Cela permet de récupérer la géométrie WKT correspondante à la ressource.
- 'FILTER ?g INTERSECTS "POLYGON((-100 20, -80 20, -80 40, -100 40, -100 20))"' : Cette clause de filtre permet de spécifier une condition spatiale en utilisant l'opération 'INTERSECTS'. Elle filtre les résultats pour ne sélectionner que les géométries ' ?g' qui ont une intersection avec le polygone donné (défini en WKT).

En résumé, cette requête SPARQL récupère les géométries WKT associées aux ressources qui sont liées à la classe "boat" dans l'ontologie Linkedgeodata et qui ont une intersection avec un polygone spécifique.

2.5 Le parsing des requetes spatiales

Le parsing d'une requête SPARQL spatiale fait référence au processus d'analyse syntaxique d'une requête SPARQL contenant des opérations spatiales pour interroger des données géospatiales.

Lors du parsing d'une requête SPARQL spatiale, les étapes de base du parsing SPARQL standard sont suivies, mais avec une prise en compte supplémentaire des opérations spatiales et des filtres spatiaux [1]. Voici les étapes du parsing d'une requête SPARQL spatiale :

1. Analyse lexicale (ou "tokenization") : La requête est divisée en "tokens" en respectant les règles de la grammaire SPARQL spatiale. Cela comprend l'identification des mots-clés, des noms de variables, des opérateurs spatiaux et des valeurs géospatiales.
2. Analyse syntaxique : Les tokens sont analysés selon les règles de la grammaire SPARQL spatiale pour déterminer la structure grammaticale de la requête. Cela implique la vérification de la conformité de la séquence de tokens par rapport aux règles syntaxiques spécifiques aux opérations spatiales.
3. Construction de l'arbre syntaxique : Un arbre syntaxique est construit pour représenter la structure hiérarchique de la requête, en tenant compte des opérations spatiales et des filtres spatiaux. Cela permet de représenter les relations et la composition des différents éléments spatiaux dans la requête.
4. Validation sémantique : Une fois que l'arbre syntaxique est construit, une vérification sémantique peut être effectuée pour s'assurer que la requête est conforme aux règles sémantiques des opérations spatiales. Cela peut inclure des vérifications telles que la validité des opérateurs spatiaux, la cohérence des types géospatiaux utilisés, etc.

Le parsing d'une requête SPARQL spatiale peut être réalisé à l'aide d'un moteur de requêtes SPARQL spatiales ou d'outils spécialisés prenant en charge les opérations spatiales. Une fois la requête correctement analysée, les informations extraites peuvent être utilisées pour exécuter la requête et interroger les données géospatiales correspondantes en appliquant les filtres spatiaux et les opérations spatiales spécifiées dans la requête.

2.6 La bibliothèque JTS

La bibliothèque JTS (Java Topology Suite) est une bibliothèque open-source écrite en Java qui fournit des fonctionnalités pour la manipulation et l'analyse des données géospatiales. Elle est largement utilisée dans le domaine des systèmes d'information géographique (SIG) et de la géoinformatique [18] .

JTS offre un ensemble complet de classes et de méthodes pour représenter, créer, modifier et analyser des objets géométriques tels que les points, les lignes, les polygones, les multipoints, les multilignes, les multipolygones, etc. La bibliothèque JTS utilise une représentation basée sur les structures de données de la topologie, permettant ainsi des opérations avancées sur les objets géométriques, notamment les opérations spatiales (intersections, unions, différences, etc.), les relations spatiales, les mesures de distance, les calculs d'aire, etc [17].

Avec cette bibliothèque l'équipe du projet RDF_QDAG notamment Mr. Houssameddine YOUSFI a implémenté deux filtres spatiaux "INTERSECTS" et "DISJOINT".

- Le filtre "INTERSECTS" est utilisé pour sélectionner des régions qui ont une relation d'intersection avec une région spécifiée. Plus précisément, il permet de récupérer les régions qui se chevauchent ou ont une intersection non vide avec une région donnée.
- Le filtre "DISJOINT" est utilisé pour sélectionner des régions qui sont totalement séparées ou qui n'ont aucune intersection avec une région spécifiée. En d'autres termes, ce filtre permet de récupérer les régions qui ne se chevauchent pas du tout avec la région donnée.

Nous avons opté sur l'utilisation du "region connection calculus" (RCC) ,en francais le "Calcul des Relations entre Régions" qui est une méthode formelle pour décrire les relations spatiales entre les régions des données du même type [16]. pour implémenter les filtres manquants.

Les relations spatiales du RCC permettent de caractériser de manière précise la structure de l'environnement, ce qui facilite la segmentation, l'analyse et la compréhension des données.

2.7 Le bulk runner

Le bulk runner est un script Python qui utilise une ligne de commande pour extraire les valeurs de configuration d'un fichier ".ini". Ces valeurs comprennent des informations telles que le chemin du dossier des requêtes, le chemin du dossier des données binaires, le chemin du fichier JAR, et bien d'autres. Ensuite il compte le nombre de plans disponibles dans un fichier de requête en utilisant une commande Java (Un plan est une stratégie ou une approche spécifique pour exécuter une requête sur les données. Pour chaque fichier de requête, il peut y avoir plusieurs plans disponibles, chacun étant une variante différente de l'exécution de la requête.). Par la suite il exécute tous les plans disponibles pour chaque fichier de requête et enregistre les résultats dans un fichier journal avec le temps d'exécution.

Globalement, ce script automatise l'exécution de requêtes et l'enregistrement des résultats en fonction des configurations spécifiées dans le fichier de configuration.

Chapitre 3

Conception

3.1 Introduction

Dans ce chapitre, nous abordons trois sujets importants. Tout d'abord, nous mettons en avant l'importance de l'abstraction dans un projet Java, en expliquant pourquoi et comment nous l'avons utilisée. Ensuite, nous explorons un patron de conception spécifique qui a été employé dans l'implémentation des entités spatiales de RDF_QDAG. Enfin, nous abordons la pratique de l'intégration continue (CI).

3.2 La création d'un filtre

Dans un projet Java orienté objet (POO), l'abstraction revêt une importance fondamentale car elle contribue à améliorer la lisibilité, la maintenabilité et la flexibilité du code. La POO permet de structurer le code en utilisant des classes et des objets qui encapsulent les données et les fonctionnalités associées. L'abstraction en POO consiste à définir des classes et des interfaces qui capturent l'essence d'un concept ou d'une entité, sans se préoccuper des détails d'implémentation. Cela permet de modéliser le système de manière plus intuitive et de favoriser la réutilisabilité du code. Grâce à l'abstraction en POO, il devient plus facile de se concentrer sur les aspects essentiels du système, ce qui facilite la compréhension du code et simplifie les modifications ultérieures.

En utilisant l'abstraction en POO, nous avons la possibilité de créer des hiérarchies de classes, où les classes de niveau supérieur définissent des caractéristiques communes, tandis que les classes de niveau inférieur fournissent des implémentations spécifiques. Cette approche nous permet de créer une structure bien organisée et modulaire, où chaque classe a un rôle clair et distinct. Grâce à cette hiérarchie, il devient plus facile de maintenir et d'étendre le code, car les modifications apportées à une classe de niveau supérieur peuvent bénéficier à toutes les classes qui en héritent. De plus, cette approche facilite également la réutilisation du code, car les classes de niveau inférieur peuvent hériter et étendre les fonctionnalités des classes de niveau supérieur. Ainsi, l'utilisation de l'abstraction en POO contribue à la création de systèmes bien structurés et évolutifs.

Une façon courante d'utiliser l'abstraction en Java est à travers les interfaces. Les interfaces fournissent un moyen de déclarer un contrat que les classes doivent respecter. En définissant des interfaces, nous spécifions les méthodes que les classes concrètes doivent implémenter, sans se soucier de leur implémentation spécifique. Cela permet de créer un code plus générique et facilite l'échange des implémentations entre différentes classes.

L'abstraction en POO facilite également la maintenance du code. En se concentrant sur les concepts et les relations entre les objets plutôt que sur les détails de l'implémentation, il devient plus facile d'apporter des modifications ou d'ajouter de nouvelles fonctionnalités sans perturber le reste du système. Cette approche permet de réduire les risques d'effets indésirables lors des modifications et simplifie la tâche des développeurs lors de la maintenance du code.

De plus, l'abstraction favorise la réutilisabilité du code. En définissant des interfaces et des classes abstraites qui capturent les comportements communs, nous pouvons créer des bibliothèques ou des modules réutilisables dans différents projets. Cela permet de gagner du temps et de réduire la duplication de code, car les fonctionnalités abstraites peuvent être utilisées et étendues dans de multiples contextes.

C'est précisément pour ces raisons que nous avons choisi d'utiliser l'abstraction dans la partie spatiale de RDF_QDAG. En utilisant des concepts abstraits et des interfaces, nous avons pu créer une structure flexible, maintenable et réutilisable pour gérer les entités spatiales dans le triplestore RDF_QDAG.

La figure 3.1 montre le diagramme de classe de la partie filtres spatiaux dans RDF_QDAG.

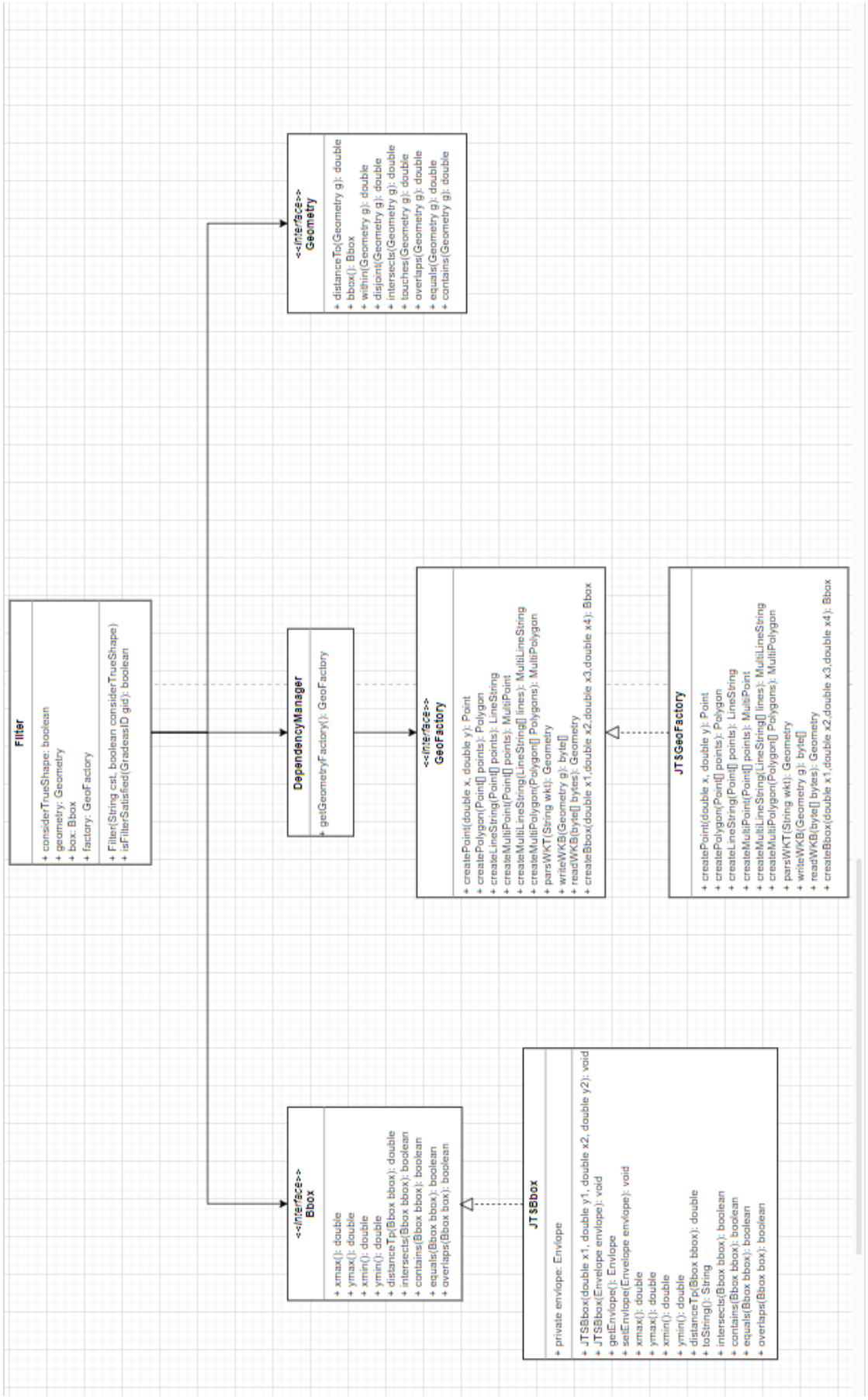


FIGURE 3.1 – Le diagramme de la partie filtres spatiaux.

3.3 Le patron de conception

Nous avons utilisé le patron de conception “Abstract Factory” dans l’implémentation des entités spatiales dans la partie spatiale de RDF_QDAG, La conception de l’Abstract Factory, ou Fabrique Abstraite, est un patron de conception couramment utilisé en Java pour créer des familles d’objets liés ou dépendants sans spécifier leur classe concrète. L’Abstract Factory fournit une interface pour créer différents types d’objets, tout en masquant les détails de leur création.

En utilisant l’Abstract Factory, nous pouvons encapsuler la logique de création d’objets dans une classe abstraite, tandis que les classes concrètes dérivées de cette abstraction implémentent la création réelle des objets. Cela permet une meilleure séparation des responsabilités et facilite la substitution des familles d’objets.

En Java, l’Abstract Factory est souvent implémenté à l’aide d’une interface abstraite qui définit les méthodes de création d’objets [3]. Les classes concrètes qui implémentent cette interface abstraite sont responsables de la création d’objets spécifiques. L’Abstract Factory permet également de garantir la cohérence des objets créés dans une famille.

La figure 3.2 montre le diagramme de classe de notre composant Geometry.

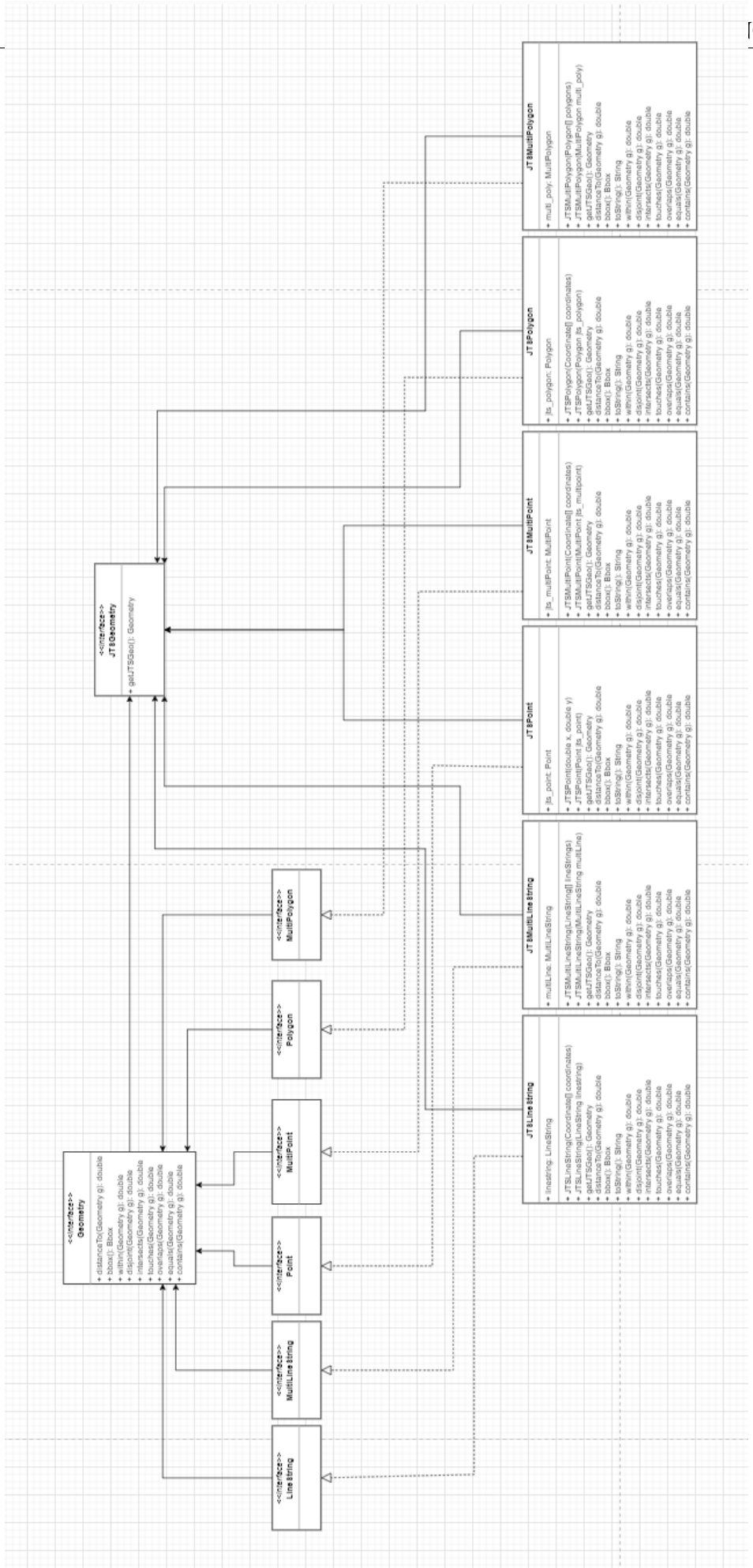


FIGURE 3.2 – Le diagramme de classe du composant Geometry

3.4 L'intégration continue

L'intégration continue (CI) est une pratique de développement logiciel qui consiste à fusionner régulièrement et automatiquement les modifications du code dans une branche principale partagée. Le processus de CI implique l'utilisation d'un système d'automatisation de build, qui compile et exécute des tests unitaires sur le code modifié à chaque nouvelle modification apportée au code source [7]. Cette pratique permet de détecter les erreurs de code plus rapidement et de manière plus fiable.

En utilisant un outil de CI, tel que GitHub Actions (https ://github.com/features/actions), nous bénéficions d'un processus continu d'intégration qui automatise la validation de notre code à chaque soumission effectuée par les développeurs. Lorsqu'un développeur soumet du code, une série de tests est exécutée automatiquement pour vérifier que ce dernier fonctionne correctement et n'entraîne pas de dysfonctionnement dans d'autres parties du système [11]. En cas d'échec des tests, le développeur est immédiatement informé de la cause du problème, facilitant ainsi la correction rapide des erreurs et l'amélioration de la qualité globale du code.

Le processus de CI offre également l'avantage d'accélérer le cycle de développement. Il permet aux développeurs de fusionner leurs modifications dans le code principal de manière rapide et régulière, favorisant ainsi une intégration continue des fonctionnalités et des correctifs. Cela réduit les risques d'incompatibilités et de conflits de fusion, tout en facilitant la collaboration au sein de l'équipe de développement.

Grâce à l'utilisation de GitHub Actions, nous bénéficions d'une machine virtuelle basée sur la dernière version d'Ubuntu sous Linux, spécialement fournie pour exécuter nos actions de CI. Cette machine fournit un environnement fiable et cohérent pour l'exécution des tests et des autres processus automatisés, garantissant ainsi des résultats précis et reproductibles.

En résumé, l'intégration continue grâce à GitHub Actions nous offre la possibilité de valider notre code de manière automatique, d'accélérer le cycle de développement et de garantir la stabilité et la qualité de notre système.

La figure 3.3 représente une démonstration du fonctionnement de l'intégration continue

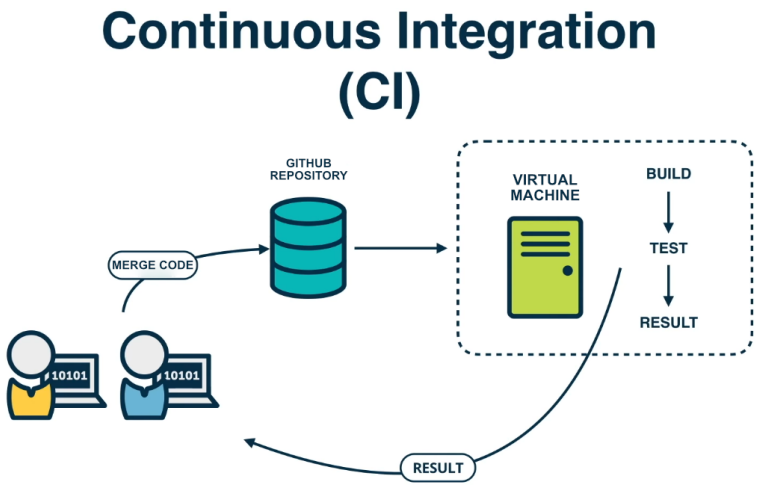


FIGURE 3.3 – Fonctionnement de l'intégration continue

Dans une machine virtuellle fournie par GitHub Actions, basée sur la dernière version d'Ubuntu sous Linux, nous exécutons une série de commandes, également appelées actions. Voici le déroulement de ces actions dans un diagramme de séquence.

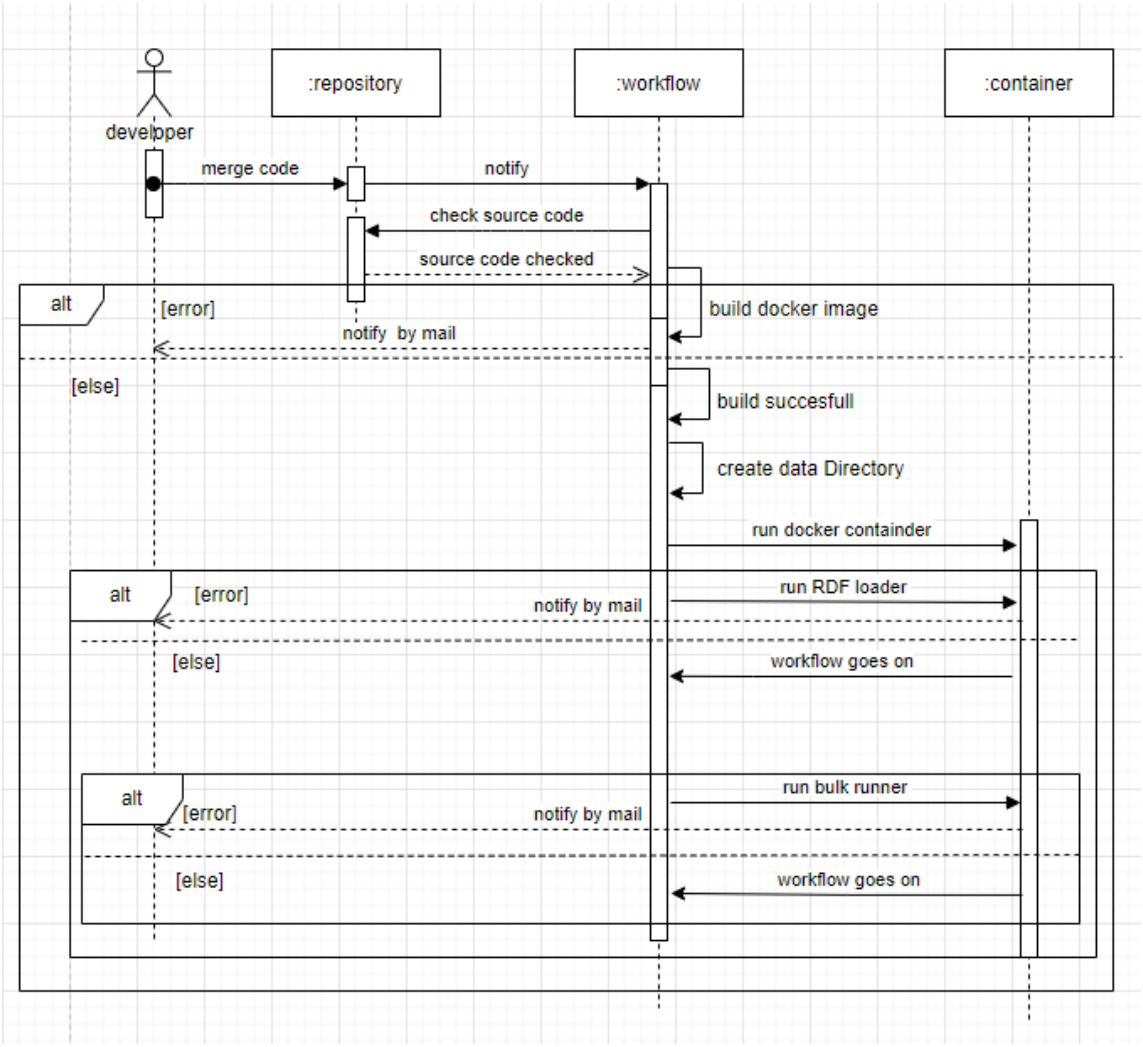


FIGURE 3.4 – Diagramme de séquence des actions

Après que le développeur ait fusionné la branche sur laquelle il travaillait avec la branche principale, nous procédons à la récupération du code source du dépôt où se trouve le pipeline (action workflow). Par la suite, une commande est exécutée pour construire une image Docker. En cas d'erreur, notre pipeline s'arrête et une notification est envoyée au développeur. Ensuite, un répertoire de données est créé, suivi du lancement d'un conteneur Docker en utilisant l'image préalablement construite. L'exécution du chargement des données dans le répertoire précédent à l'intérieur du conteneur Docker est alors lancée. Enfin, le script Bulk Runner est exécuté dans notre conteneur Docker. Chaque étape est exécutée séquentiellement dans l'ordre spécifié. Une fois que toutes les étapes sont terminées, le job est considéré comme terminé.

Il est essentiel de prendre en compte plusieurs aspects concernant les limites de la machine virtuelle fournie par GitHub Actions :

— Temps d'exécution : Le temps maximum d'exécution par défaut pour une action GitHub est de 6 heures. Cela signifie que si votre workflow dépasse cette limite, il sera arrêté.

— Mémoire : La quantité maximale de mémoire disponible dépend du type d'environnement. Par exemple, pour les environnements Linux, le maximum est généralement de 3,75 Go pour les machines virtuelles hébergées et de 7 Go pour les machines virtuelles exécutées sur des hôtes dédiés. Les environnements Windows et macOS peuvent avoir des limites différentes.

— Stockage temporaire : La machine virtuelle dispose d'un espace de stockage temporaire limité pour les fichiers et les artefacts générés lors de l'exécution de vos workflows. L'espace de stockage temporaire est généralement de 10 Go, mais cela peut varier en fonction de l'environnement.

— Puissance de calcul : Les machines virtuelles utilisées dans GitHub Actions sont basées sur des ressources partagées. Par conséquent, les performances peuvent varier en fonction de la charge du système et du nombre d'actions en cours d'exécution simultanément.

Chapitre 4

Réalisation

4.1 Introduction

Dans ce chapitre, nous abordons la concrétisation de notre approche, en mettant l'accent sur l'implémentation des filtres spatiaux. Nous débutons en présentant comment ces filtres sont mis en œuvre. Ensuite, nous examinons en détail un exemple de filtre spécifique. Enfin, nous discutons de l'importance des tests unitaires dans ce contexte et nous allons présenter un exemple sur le bulk runner.

4.2 Implémentation des filtres

Les filtres spatiaux sont basés sur le "region connection calculus" (RCC) ,en français le "Calcul des Relations entre Régions" qui est une méthode formelle pour décrire les relations spatiales entre les régions des données du même type [16]. Les relations spatiales du RCC permettent de caractériser de manière précise la structure de l'environnement, ce qui facilite la segmentation, l'analyse et la compréhension des données.

4.2.1 Le filtre Within

Le filtre "within" signifie que la première région est complètement contenue dans la seconde région. Plus précisément, cela signifie que chaque point de la première région est également contenu dans la seconde région et ne déborde pas de ses limites. Il est utilisé pour trouver des régions qui se trouvent à l'intérieur d'une région donnée.

4.2.2 Le filtre Touches

Le filtre "touches" signifie que les frontières de deux régions se touchent sans pour autant se recouvrir. Autrement dit, il y a un contact ponctuel ou linéaire entre les deux régions, mais elles ne se chevauchent pas. Il est utilisé pour trouver des régions qui ont des frontières communes, mais qui ne se recouvrent pas.

4.2.3 Le filtre Overlaps

Le filtre "overlaps" signifie que les deux régions ont une partie en commun, c'est-à-dire qu'elles se chevauchent. Plus précisément, il y a au moins un point qui appartient à la fois à la première région et à la seconde région. Il est utilisé pour détecter si deux objets se recouvrent ou se superposent, ou pour trouver des zones qui sont partiellement recouvertes par plusieurs régions.

4.2.4 Le filtre Contains

Le filtre "contains" signifie que la première région contient entièrement la seconde région. Plus précisément, cela signifie que chaque point de la seconde région est également contenu dans la première région. Il est utilisé pour trouver des régions qui sont entièrement contenues dans une région donnée.

4.2.5 Le filtre Equals

Le filtre “equals” signifie que les deux régions sont identiques, c'est-à-dire qu'elles ont exactement les mêmes points et que leurs frontières sont identiques. Il est utilisé pour détecter si deux régions sont exactement les mêmes, ou pour trouver des régions qui ont des frontières identiques.

Il est important de noter que la relation "equals" est assez stricte et rarement observée en pratique.

En effet, les régions peuvent souvent avoir des variations mineures dans leur forme ou leur position, ce qui peut les rendre légèrement différentes. Dans ces cas-là, d'autres relations spatiales telles que "overlaps" ou "contains" peuvent être plus appropriées pour décrire la relation entre les régions.

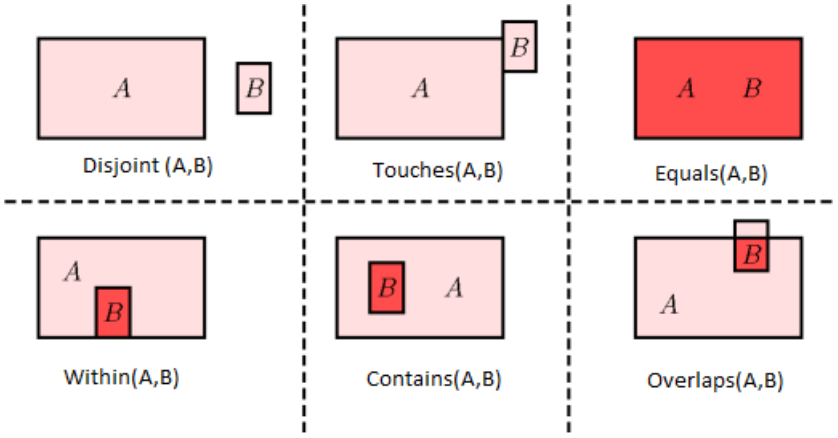


FIGURE 4.1 – Représentation géométrique des filtres

4.3 Exemple de l'implémentation du filtre 'Overlaps'

La classe OverlapsFilter qui implémente les interfaces "Filter" et "SpatialFilter" contient un constructeur ainsi qu'une méthode de type booléen qui renvoie true si les conditions du filtre sont satisfaites.

Voici un aperçu du constructeur.

```
public class OverlapsFilter implements Filter,SpatialFilter {
    private boolean considerTrueShape;

    private Geometry geometry;
    private Bbox box;

    /**
     * Construct the filter
     *
     * @param cst              The wkt to parse and check if shapes intersect
     * @param considerTrueShape True if the filter should consider true shapes
     *                          and False if it should consider only the bounding
     *                          boxes
     */
    public OverlapsFilter(String cst, boolean considerTrueShape) {
        GeoFactory factory = DependencyManager.getGeometryFactory();
        this.considerTrueShape = considerTrueShape;
        try {
            geometry = factory.parseWKT(cst);
            box = geometry.bbox();
        } catch (WKTParsingException ex) {
            Logger.getLogger(OverlapsFilter.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
```

FIGURE 4.2 – Constructeur de la classe Overlaps

Déclaration des variables de la classe :

— "considerTrueShape" : un booléen qui indique si le filtre doit considérer les formes réelles (true) ou seulement les boîtes englobantes (bounding boxes) (false).

— "geometry" : une instance de la classe "Geometry" qui représente la forme géométrique.

— "box" : une instance de la classe "Bbox" qui représente la boîte englobante de la forme géométrique (C'est le rectangle minimal qui englobe la forme).

Constructeur de la classe :

— Le constructeur prend deux paramètres : "cst" (une chaîne de caractères représentant une forme géométrique au format WKT) et "considerTrueShape" (un booléen).

— Il récupère une instance de la classe "GeoFactory" à partir du gestionnaire de dépendances (DependencyManager).

— Il assigne la valeur de "considerTrueShape" à la variable correspondante de la classe.

— Il parse la chaîne "cst" en une instance de la classe "Geometry" en utilisant la fabrique de géométrie (factory) et l'assigne à la variable "geometry".

— Il calcule la boîte englobante de la forme géométrique et l'assigne à la variable "box".

— En cas d'erreur lors du parsing de la chaîne WKT, une exception de type "WKTParsingException" est levée et affichée dans les logs.

En résumé, ce code initialise un filtre "OverlapsFilter" avec une forme géométrique spécifiée en WKT et une option pour indiquer si le filtre doit considérer les formes réelles ou seulement les boîtes englobantes.

Voici un aperçu sur la méthode isFilterSatisfied :

```
public boolean isFilterSatisfied(GradeasID gid) {
    if (gid.getGeoData() == null)
        return false;

    //Filter step just using the MBR
    Bbox entryBox = gid.getBbox();
    if (!entryBox.overlaps(this.box)) {
        return false;
    }

    //if user don't want to check the true shapes we return true
    if (!this.considerTrueShape) {
        return true;
    }

    // If the mbr is a point we return true (no need for true shape since it's a point
    if (entryBox.xmin() == entryBox.xmax() && entryBox.ymin() == entryBox.ymax()) {
        return true;
    }

     // Refignment step (considering real chape of the object)
     Geometry entryGeo = gid.getGeoData();


     if (!entryGeo.overlaps(this.geometry)) {
         return false;
     }

    return true;
}
```

FIGURE 4.3 – La méthode isFilterSatisfied

Ce code représente une méthode de la classe "OverlapsFilter" appelée "isFilterSatisfied" qui prend en paramètre un objet de type "GradeasID" et retourne un booléen. Voici ce que fait cette méthode :

1. Vérification si "geoData" de l'objet "gid" est nul :
   - Si "geoData" est nul, la méthode retourne false, indiquant que le filtre n'est pas satisfait.
2. Étape de filtrage en utilisant la boîte englobante (Bbox) :
   - La méthode récupère la boîte englobante (Bbox) de l'objet "gid" et l'assigne à la variable "entryBox".
   - Si la boîte englobante "entryBox" ne chevauche pas la boîte englobante du filtre en cours (this.box), la méthode retourne false, indiquant que le filtre n'est pas satisfait.
3. Vérification si les vraies formes doivent être prises en compte :
   - Si la variable "considerTrueShape" du filtre est à false, la méthode retourne true, indiquant que le filtre est satisfait sans considérer les vraies formes.
4. Vérification si la boîte englobante est un point :
   - Si les coordonnées xmin et xmax de la boîte englobante "entryBox" sont égales, ainsi que les coordonnées ymin et ymax, cela signifie que la boîte englobante représente un point.
   - Dans ce cas, la méthode retourne true, indiquant que le filtre est satisfait, car il n'est pas nécessaire de vérifier les vraies formes pour un point.
5. Étape de raffinement (en considérant la vraie forme de l'objet) :
   - La méthode récupère la vraie forme géométrique (Geometry) de l'objet "gid" et l'assigne à la variable "entryGeo".
   - Si la vraie forme géométrique "entryGeo" ne chevauche pas la forme géométrique du filtre en cours (this.geometry), la méthode retourne false, indiquant que le filtre n'est pas satisfait.
6. Si toutes les conditions précédentes sont satisfaites, la méthode retourne true, indiquant que le filtre est satisfait.

En résumé, cette méthode vérifie si un objet de type "GradeasID" satisfait les conditions du filtre "OverlapsFilter" en considérant la boîte englobante et éventuellement la vraie forme géométrique de l'objet.

4.4 Les tests unitaires des filtres

4.4.1 Introduction

Les tests unitaires en Java sont des morceaux de code écrits pour vérifier le bon fonctionnement des unités individuelles d'un programme [4]. Ces tests sont conçus pour valider le comportement attendu des méthodes, des classes ou même de petites parties de code. Les tests unitaires offrent de nombreux avantages. Tout d'abord, ils permettent de détecter rapidement les erreurs et les bugs, facilitant ainsi le processus de débogage. En écrivant des tests pour chaque fonctionnalité, les développeurs peuvent s'assurer que les modifications ultérieures n'ont pas d'effets indésirables sur les fonctionnalités existantes.

En outre, les tests unitaires servent également de documentation vivante, car ils fournissent des exemples clairs de l'utilisation des différentes parties du code. En somme, les tests unitaires en Java permettent d'améliorer la fiabilité, la maintenabilité et la lisibilité du code, ce qui conduit à un développement plus efficace et à des logiciels de meilleure qualité.

4.4.2 Exemple détaillé

Nous avons pris le test de 'EqualsFilter' comme exemple :

```
public class EqualsFilterTest {

    GradeasID gid = Mockito.mock(classToMock:GradeasID.class);

    JTSBbox bbox = new JTSBbox(x1:10, y1:10, x2:40, y2:40);

    Coordinate p1 = new Coordinate(x:30, y:10);
    Coordinate p2 = new Coordinate(x:40, y:40);
    Coordinate p3 = new Coordinate(x:20, y:40);
    Coordinate p4 = new Coordinate(x:10, y:20);

    Coordinate coordinates[] = { p1, p2, p3, p4, p1 };

    JTSPolygon polygon = new JTSPolygon(coordinates);
```

FIGURE 4.4 – EqualsFilterTest

Initialisation des variables et objets nécessaires pour les tests :

— La classe utilise la bibliothèque ‘Mockito’ pour créer un objet simulé de la classe "GradeasID" appelé "gid".

— Un objet "JTSBbox" appelé "bbox" est créé avec des coordonnées spécifiques.

— Quatre objets "Coordinate" sont créés avec des coordonnées spécifiques (p1, p2, p3, p4).

— Un tableau de coordonnées "coordinates" est créé en utilisant les objets "Coordinate" précédemment définis.

— Un objet "JTSPolygon" appelé "polygon" est créé en utilisant le tableau de coordonnées.

Ensuite nous testons lorsque l’objet "GradeasID" passé en paramètre a une valeur nulle pour la propriété "geoData".

```
@Test
public void TestGradeasID_null() {

    EqualsFilter equalsFilter = new EqualsFilter(cst:"POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))", considerTrueShape:true);

    when(gid.getGeoData()).thenReturn(value:null);

    assertEquals(equalsFilter.isFilterSatisfied(gid), actual:false);
}
```

FIGURE 4.5 – Test du ‘GradeasID’

Nous testons par la suite si la boîte englobante (Bbox) de l’objet "GradeasID" n’est pas égale à la Bbox de la forme géométrique définie par le filtre

```
@Test
public void Bbox_not_equals() {

    EqualsFilter equalsFilter = new EqualsFilter(cst:"POLYGON ((30 10, 50 40, 20 40, 10 20, 30 10))", considerTrueShape:true);

    when(gid.getGeoData()).thenReturn(polygon);
    when(gid.getBbox()).thenReturn(bbox);

    assertEquals(equalsFilter.isFilterSatisfied(gid), actual:false);
}
```

FIGURE 4.6 – Test Bbox égales

Nous testons par la suite si la boîte englobante (Bbox) de l'objet "GradeasID" est égale à la Bbox de la forme géométrique définie par le filtre sans la vraie forme.

```
@Test
public void TestEquals_without_true_shape() {
    EqualsFilter equalsFilter = new EqualsFilter(cst:"POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))", considerTrueShape:false);

    when(gid.getGeoData()).thenReturn(polygon);
    when(gid.getBbox()).thenReturn(bbox);

    assertEquals(equalsFilter.isFilterSatisfied(gid), actual:true);
}
```

FIGURE 4.7 – Test Bbox non-égales

Nous testons par la suite quand la Bbox est égale a la Bbox de la forme géométrique définie par le filtre mais la vraie forme n'est pas égale à la forme géométrique définie par le filtre.

```
@Test
public void Test_not_Equals_with_true_shape() {
    EqualsFilter equalsFilter = new EqualsFilter(cst:"POLYGON ((40 10, 40 40, 20 40, 10 20, 40 10))", considerTrueShape:true);

    when(gid.getGeoData()).thenReturn(polygon);
    when(gid.getBbox()).thenReturn(bbox);

    assertEquals(equalsFilter.isFilterSatisfied(gid), actual:false);

    EqualsFilter equalsFilter1 = new EqualsFilter(cst:"POLYGON ((40 10, 40 40, 20 40, 10 20, 40 10))", considerTrueShape:false);
    assertEquals(equalsFilter1.isFilterSatisfied(gid), actual:true);
}
```

FIGURE 4.8 – Test sans vraie forme

Enfin nous vérifions que la forme géométrique de l'objet "GradeasID" est égale à la forme géométrique définie par le filtre.

```
@Test
public void TestEquals_with_true_shape() {
    EqualsFilter equalsFilter = new EqualsFilter(cst:"POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))", considerTrueShape:true);

    when(gid.getGeoData()).thenReturn(polygon);
    when(gid.getBbox()).thenReturn(bbox);

    assertEquals(equalsFilter.isFilterSatisfied(gid), actual:true);

    EqualsFilter equalsFilter1 = new EqualsFilter(cst:"POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))", considerTrueShape:false);
    assertEquals(equalsFilter1.isFilterSatisfied(gid), actual:true);
}
```

FIGURE 4.9 – Test avec vraie forme

4.5 Le fichier log du bulk runner

Dans la dernière partie du chapitre 2, nous avons abordé le script "bulk runner", conçu pour automatiser l'exécution de requêtes et l'enregistrement des résultats selon les configurations spécifiées dans le fichier de configuration. L'enregistrement des résultats est effectué dans un fichier ".log" qui est stocké dans le dossier de logs. Voici un aperçu du contenu de ce fichier :

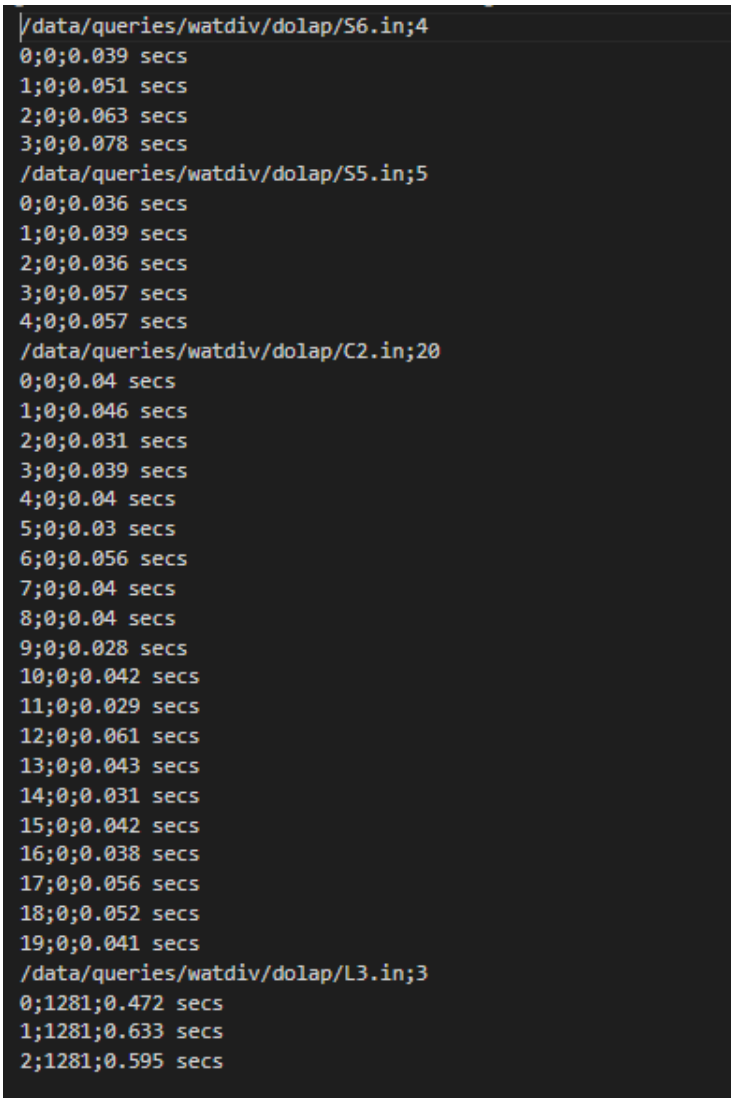


FIGURE 4.10 – Exemple d'un fichier log

Pour analyser en détail le contenu du fichier, prenons l'exemple suivant : "/data/queries /watdiv/dolap/S6.in", suivi du nombre "4". Ce chemin indique le fichier de requête utilisé, tandis que le "4" représente le nombre de plans associés à cette requête.

Ensuite, chaque ligne du fichier contient trois paramètres, représentés par une séquence de nombres séparés par des points-virgules. Ces paramètres correspondent à l'indexation, au nombre de résultats et au temps d'exécution pour chaque plan spécifique.

L'objectif de notre démarche est de vérifier si le nombre de résultats pour chaque plan d'une requête est identique. Dans ce but, nous avons ajouté une condition dans le script "bulk runner". Si nous constatons que les résultats diffèrent entre les plans, l'exécution est interrompue, ce qui indique un problème quelque part dans le code de RDF_QDAG. L'interruption de l'exécution aura un impact sur le déroulement des actions de l'intégration continue en cours, car l'exécution de ce script en fait partie. Afin de garantir une notification rapide du développeur, un e-mail automatique lui sera envoyé pour le tenir informé de la situation. Ainsi, il pourra prendre les mesures nécessaires pour résoudre le problème, effectuer les corrections nécessaires et fusionner à nouveau son code avec la branche principale du projet.

Chapitre 5

Gestion de projet

5.1 Introduction

Dans ce chapitre, nous allons voir la manière utilisée pour gérer notre projet notamment la planification des taches, le modèle de développement ainsi que les différents outils collaboratifs utilisés.

5.2 Outils collaboratifs

5.2.1 Overleaf

Overleaf [a] est un éditeur LATEX en ligne et gratuit, il est utilisé pour rédiger des articles et des rapports scientifiques. Nous avons choisi d'utiliser la plateforme overleaf pour les raisons suivantes :

- Avoir la possibilité de rédiger de manière collaborative.
- La flexibilité dans la structure du rapport.
- Avoir la possibilité de visualiser le rapport en PDF

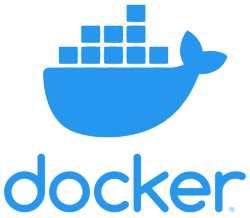
a. https ://www.overleaf.com/

5.2.2 GitHub

GitHub [a] est une plate-forme qui permet aux développeurs de travailler sur le même projet en même temps et facilement, et aussi permet de stocker leurs projets avec ses différentes versions. Nous avons utilisé cette plateforme pour faciliter le travail en binôme et pouvoir sauvegarder les différentes étapes de l'évolution de notre projet.

a. https ://github.com/

5.2.3 Docker



Docker est une plateforme logicielle qui permet de créer, déployer et exécuter des applications de manière isolée dans des conteneurs. Ces conteneurs fournissent un environnement léger et portable, dans lequel les applications peuvent fonctionner de manière cohérente quel que soit le système d'exploitation sous-jacent. Le principal objectif de Docker est de simplifier le déploiement et la gestion des applications. Il vise à résoudre les défis et les complexités liés à la mise en place d'un environnement d'exécution cohérent pour les applications, que ce soit en développement, en test ou en production.

5.2.4 Microsoft Teams

Microsoft Teams [a] est un système lancé par Microsoft en novembre 2016, centré sur la communication audio et vidéo qui s'adresse essentiellement aux entreprises professionnelles et établissements éducatifs. Cette plateforme offre une interface graphique accessible, ainsi que différentes options d'utilisation comme le partage d'écran, l'enregistrement des sessions et le transfère / la sauvegarde des fichiers. Nous avons utilisé cet outil pour faire nos réunions de projet.

a. https ://teams.microsoft.com/

5.3 Modèle de développement et planning

La méthode Agile Scrum [4] est un cadre de gestion de projet itératif et incrémental largement utilisé dans le développement de logiciels. Son objectif principal est de fournir une approche flexible et adaptative pour la gestion des projets, en favorisant la collaboration, la transparence et l'amélioration continue.

Scrum se base sur des cycles de développement courts appelés "itérations" ou "sprints", généralement d'une durée de 1 à 4 semaines. Chaque sprint démarre par une réunion de planification où l'équipe définit les objectifs et les livrables attendus. L'équipe de développement travaille ensuite de manière autonome pour réaliser les tâches assignées pendant le sprint.

À la fin de chaque sprint, l'équipe de développement présente les résultats obtenus lors d'une démonstration ou d'une revue. Les commentaires et les retours obtenus sont pris en compte pour améliorer et ajuster le plan du prochain sprint.

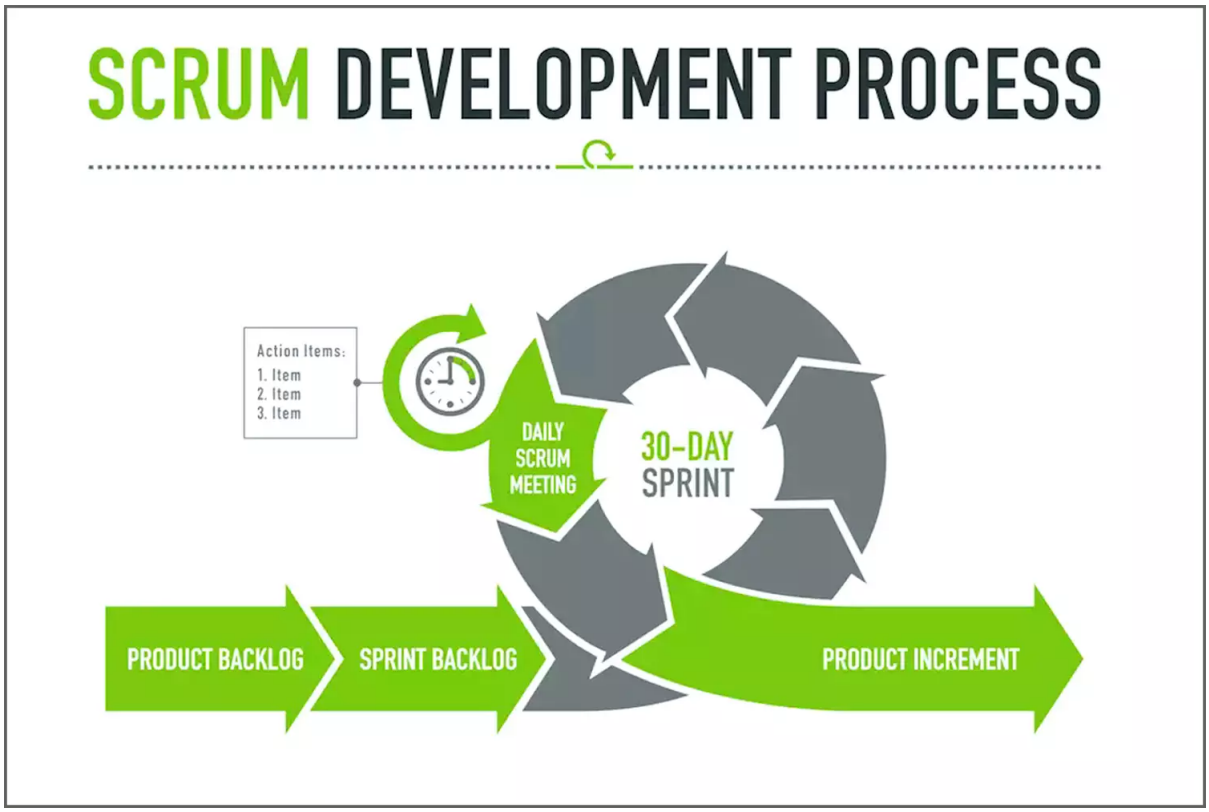


FIGURE 5.1 – Schéma méthode agile

— Sprint 1 (1er février-1er mars)

Nous avons consacré notre premier sprint à l'intégration et à la compréhension du projet, nous nous sommes réunis pour analyser les exigences du projet, comprendre les objectifs et les besoins, ainsi que pour installer les outils nécessaires à notre travail. Nous avons également établi une communication claire et ouverte entre tous les membres de l'équipe, afin de garantir une compréhension commune du projet.

— Sprint 2 (1er mars-10 avril)

Pour le deuxième sprint , notre objectif principal était d'apprendre les outils essentiels à notre projet, notamment Docker pour la gestion des conteneurs, l'exécution de requêtes et le chargement des données. Nous avons également commencé à explorer les différentes manières dont nous pouvons utiliser ces technologies dans notre projet, en mettant l'accent sur la performance et l'efficacité.

— Sprint 3 (10 avril-1er mai)

Pour ce sprint nous nous sommes concentrés sur la logique et l'implémentation des filtres. Après avoir acquis une compréhension solide sur la logique de Mr. Houssameddine YOUSFI nous avons travaillé sur l'implémentation des filtres spatiaux.

— Sprint 4 (1er mai-18 mai)

Pendant ce sprint, notre attention s'est portée sur l'implémentation des tests unitaires. Nous avons adopté une approche rigoureuse en développant des cas de test complets pour chaque fonctionnalité et filtre implémenté. Les tests unitaires nous ont permis de vérifier la qualité et la fiabilité de notre code, en identifiant et en corrigeant rapidement les erreurs éventuelles. Nous avons également effectué des ajustements et des améliorations en fonction des résultats des tests.

— sprint 5 (18 mai-10 juin)

Pour le dernier sprint, notre objectif était de corriger les bugs identifiés lors des tests précédents et de nous assurer que toutes les requêtes fonctionnaient correctement avec les filtres implémentés. Nous avons consacré du temps à l'identification, à la résolution et à la vérification des problèmes. Nous avons également effectué des tests complets pour vérifier que toutes les fonctionnalités fonctionnaient sans problème.

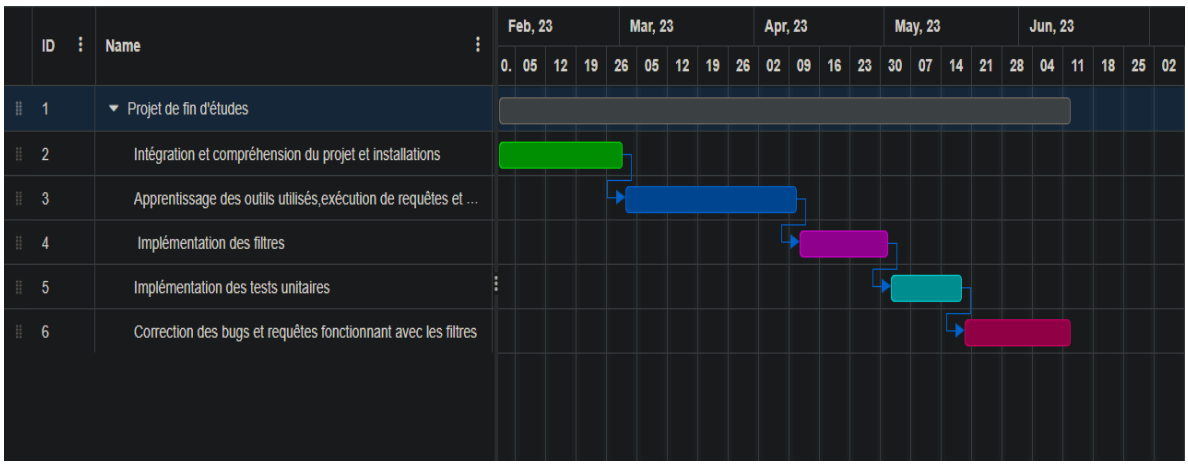


FIGURE 5.2 – Diagramme de Gantt

En suivant cette approche en sprints, nous avons pu avancer de manière itérative et progressive dans notre projet. Cette méthodologie nous a permis de rester concentrés, de résoudre rapidement les problèmes et d'assurer une progression efficace vers notre objectif final.

5.4 Conclusion

Dans ce chapitre, nous avons vu comment nous avons pu gérer notre projet, nous avons vu les différents outils collaboratifs utilisés afin de faciliter cette tâche. Nous avons également parlé de la méthode agile (scrum) et nous avons conclu ce chapitre par un diagramme de Gantt pour illustrer le temps alloué pour chaque sprint.

Chapitre 6

Conclusion et perspectives

6.1 Conclusion

Notre projet de fin d'étude est le fruit d'une collaboration entre le laboratoire LRIT de l'université de Tlemcen et le laboratoire LIAS de l'université de Poitiers. L'objectif de notre projet était de fournir un cadre pratique pour la partie spatiale de RDF_QDAG. Nous avons particulièrement concentré nos efforts sur les filtres spatiaux manquants et avons développé une approche novatrice pour automatiser les workflows de développement logiciel en utilisant GitHub Actions.

De plus, nous avons apporté des modifications à l'analyse syntaxique des requêtes SPARQL afin d'intégrer les filtres spatiaux dans ces requêtes. Cette intégration permet de tirer pleinement parti des données spatiales, qui sont des informations liées à une position géographique spécifique. Ces données, généralement représentées sous forme de coordonnées géographiques, jouent un rôle essentiel dans de nombreux domaines et applications, tels que la planification urbaine, la cartographie des infrastructures urbaines, la surveillance des changements climatiques, l'analyse de la déforestation, le suivi des migrations animales, et la prévision des risques naturels tels que les inondations et les feux de forêt.

En outre l'interconnexion massive des données, des informations et des connaissances a permis l'émergence des graphes de connaissances, qui apportent une sémantique pour contextualiser les données et établir des relations entre elles. Le Resource Description Framework (RDF) et SPARQL se sont avérés être des outils pertinents pour représenter et interroger ces données. En combinant ces technologies avec notre approche novatrice des filtres spatiaux, notre projet contribue à l'intégration, l'unification, l'analyse et le partage des données spatiales dans le contexte des graphes de connaissances.

Pour conclure, notre projet de fin d'étude représente une avancée significative dans le domaine de l'intégration des données spatiales avec RDF_QDAG. Les résultats obtenus ouvrent de nouvelles perspectives pour l'utilisation et l'exploitation des données spatiales dans divers domaines d'application. Nous espérons que notre travail servira de base solide pour de futures recherches et développements visant à améliorer encore davantage l'intégration des données spatiales dans les graphes de connaissances.

6.2 Perspectives

Ce travail offre différentes perspectives pour améliorer la partie spatiale de RDF_QDAG. Deux pistes de développement ont été envisagées :

1. La première perspective est d'effectuer une étude sur l'évaluation des jointures spatiales en utilisant une stratégie adaptée. Cette exploration vise à développer des techniques et des méthodes spécifiques pour optimiser les opérations de jointure spatiale, en prenant en compte les caractéristiques et les contraintes propres à RDF_QDAG. Cette démarche pourrait conduire à des améliorations significatives en termes de performances et d'efficacité des requêtes spatiales.

2. La deuxième perspective consiste à charger une grande base de données lors de l'intégration continue (CI). Pour ce faire, nous pouvons remplacer GitHub par GitLab afin d'héberger les actions sur une machine puissante, garantissant ainsi un processus d'intégration et de déploiement efficace à grande échelle. Cette approche nous permet de tester les requêtes à travers le script "bulk runner" sur une grande base de données, ce qui peut parfois révéler des problèmes qui ne se manifestent pas sur une petite base.

En somme, ces perspectives ouvrent des opportunités d'amélioration de RDF_QDAG sur le plan de l'intégration continue et de la prise en compte de la jointure spatiale, offrant ainsi de nouvelles possibilités de développement et de recherche dans ce domaine.

Bibliographie

[1] Andreas Brodt, Daniela Nicklas, and Bernhard Mitschang. Deep integration of spatial query processing into native rdf triple stores. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 33–42, 2010.

[2] I Budak Arpinar, Amit Sheth, Cartic Ramakrishnan, E Lynn Usery, Molly Azami, and Mei-Po Kwan. Geospatial ontology development and semantic analytics. *Transactions in GIS*, 10(4) :551–575, 2006.

[3] Aleksandar Bulajic and Slobodan Jovanovic. An approach to reducing complexity in abstract factory design pattern. *Journal of Emerging Trends in Computing and Information Sciences*, 3(10), 2012.

[4] H Frank Cervone. Understanding agile project management methods using scrum. *OCLC Systems & Services : International digital library perspectives*, 27(1) :18–22, 2011.

[5] Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin. *The Semantic Web–ISWC 2017 : 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I*, volume 10587. Springer, 2017.

[6] Sidi Mohamed El Amraoui, Mohamed Rouchdi, Mourad Bouziani, and Abdelwahed El Idrissi. Integration du sig et de l'analyse hierarchique multicritere pour l'aide dans la planification urbaine : Etude de cas de la province de khemisset, maroc. *Papeles de Geografía*, (63), 2017.

[7] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.

[8] Jose Manuel Gomez-Perez, Jeff Z Pan, Guido Vetere, and Honghan Wu. Enterprise knowledge graph : An introduction. In *Exploiting linked data and knowledge graphs in large organisations*, pages 1–14. Springer, 2017.

[9] Efdal Kaya, Muge Agca, Fatih Adiguzel, and Mehmet Cetin. Spatial data analysis with r programming for environment. *Human and ecological risk assessment : An International Journal*, 25(6) :1521–1530, 2019.

[10] Abdallah Khelil, Amin Mesmoudi, Jorge Galicia, Ladjel Bellatreche, Mohand-Saïd Hacid, and Emmanuel Coquery. Combining graph exploration and fragmentation for scalable rdf query processing. *Information Systems Frontiers*, 23 :165–183, 2021.

[11] Timothy Kinsman, Mairieli Wessel, Marco A Gerosa, and Christoph Treude. How do software developers use github actions to automate their workflows ? In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 420–431. IEEE, 2021.

[12] Krys J Kochut and Maciej Janik. Sparqler : Extended sparql for semantic association discovery. In *The Semantic Web : Research and Applications : 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007. Proceedings 4*, pages 145–159. Springer, 2007.

[13] Dave Kolas. Supporting spatial semantics with sparql. *Transactions in GIS*, 12 :5–18, 2008.

[14] Sameh K Mohamed, Aayah Nounu, and Vít Nováček. Biological applications of knowledge graph embedding models. *Briefings in bioinformatics*, 22(2) :1679–1693, 2021.

[15] Edzer J Pebesma et al. Simple features for r : standardized support for spatial vector data. *R J.*, 10(1) :439, 2018.

[16] Jochen Renz and Bernhard Nebel. On the complexity of qualitative spatial reasoning : A maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1-2) :69–123, 1999.

[17] Thomas Schwarz, Nicola Hoenle, Matthias Grossmann, and Daniela Nicklas. A library for managing spatial context using arbitrary coordinate systems. In *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second*, pages 48–52. IEEE, 2004.

[18] Bernhard Seeger and Peter Widmayer. Geographic information systems. In *Handbook of Data Structures and Applications*, pages 871–888. Chapman and Hall/CRC, 2018.

[19] Mingxiong Zhao, Han Wang, Jin Guo, Di Liu, Cheng Xie, Qing Liu, and Zhibo Cheng. Construction of an industrial knowledge graph for unstructured chinese text learning. *Applied Sciences*, 9(13) :2720, 2019.

Table des figures

2.1 Rprésentation des entités du WKT 7
2.2 Requête SPARQL spatial 8

3.1 Le diagramme de la partie filtres spatiaux. 16
3.2 Le diagramme de classe du composant Geometry 18
3.3 Fonctionnement de l'intégration continue 20
3.4 Diagramme de séquence des actions 21

4.1 Représentation géométrique des filtres 25
4.2 Constructeur de la classe Overlaps 26
4.3 La méthode isFilterSatisfied 27
4.4 EqualsFilterTest 29
4.5 Test du 'GradeasID' 30
4.6 Test Bbox égales 30
4.7 Test Bbox non-égales 31
4.8 Test sans vraie forme 31
4.9 Test avec vraie forme 32
4.10 Exemple d'un fichier log 33

5.1 Schéma méthode agile 38
5.2 Diagramme de Gantt 39

Table des abréviations

Abréviation	Signification
Bbox	Bounding box
CI	Continuous Integration
JTS	Java Topology Suite
POO	Programmation Orientée objet
QDAG	Querying Data As Graphs
RCC	Region connection calculus
RDF	Resource Description Framework
SIG	Systèmes d'Information Géographique
SPARQL	Simple Protocol and Rdf Query Language
WKT	Well-known Text

Résumé

Afin de finaliser notre Master en Génie Logiciel, nous avons intégré l'équipe du laboratoire LIAS à Poitiers dans le but d'implémenter des filtres spatiaux au sein des triplestores de RDF_QDAG. Pour ce faire, nous avons utilisé le patron de conception de l'Abstract Factory et réussi à mettre en place des tests unitaires pour les valider. De plus, nous avons automatisé les tests grâce à l'utilisation de GitHub Actions, ce qui nous permet de les exécuter automatiquement chaque fois qu'un développeur ajoute ou modifie le code source.

Mots-clès : Graphe de connaissance, Langage SPARQL, Triplestores, Optimisation, Données spatio-temporelles.

Abstract

In order to complete our Master's degree in Software Engineering, we joined the LIAS laboratory team in Poitiers with the objective of implementing spatial filters within RDF_QDAG triplestores. To achieve this, we employed the Abstract Factory design pattern and successfully established unit tests to validate them. Furthermore, we automated the tests using GitHub Actions, enabling automatic execution whenever a developer adds or modifies the source code.

Keywords : Knowledge Graph, SPARQL Language, Triplestores, Optimization, Spatiotemporal Data.

ملخص

من أجل إكمال درجة التخرج في هندسة البرمجيات ، انضممنا إلى فريق مخبر LIAS في Poitiers بهدف تنفيذ المرشحات المكانية داخل RDF_QDAG triplestores . لتحقيق ذلك ، استخدمنا نمط تصميم Abstract Factory وأسسنا بنجاح اختبارات الوحدة للتحقق من صحتها. علاوة على ذلك ، قمنا بأتمتة الاختبارات باستخدام Github Actions ، مما يتيح التنفيذ التلقائي كلما أضاف المطور أو يعدل الشفرة المصدرية.

الكلمات المفتاحة :

الرسم البياني ، لغة SPARQL ، Triplestores ، تحسين ، بيانات المكان والزمان.