Analyse des techniques de stockage et d'interrogation du web sémantique :
Application à la médiation de métadonnées environnementales

Rapport de Projet de Fin d'Etudes (PFE)

Novembre 2012

EV2 CHAPEAU, EV2 DROZ-BARTHOLET, EN10

Remerciements

Nous tenons à remercier l'UMR Espace Dev pour nous avoir accueilli dans sa structure et nous avoir offert cette opportunité de stage.

Nous remercions également chaleureusement nos tuteurs de stage Jean-Christophe Desconnets et Christelle Pierkot pour leur disponibilité, leur soutien et leurs relectures attentives, critiques et orthographiques qui se sont avérées bien plus que profitables. Nous remercions également Thérèse Libourel pour son accueil en début de stage et ses précieux conseils tant au niveau de la modélisation que de la forme de notre rapport.

Nous remercions Isabelle Tores pour son aide et sa patience lors des démarches administratives qui se sont parfois révélées compliquées.

Nous remercions Marie Angélique Laporte et Samuel Andres pour leur bonne humeur et leurs lumières sur nombre de sujets qui seraient restés obscures. Nous n'oublions pas non plus Bertrand Guerrero et sa jovialité.

Nous remercions l'ensemble des membres de l'équipe SIC de l'IRD qui se sont données la peine de venir assister à nos différentes présentations durant le stage, nous apportant critiques, remarques mais également motivation au travers de leur intérêt. Nous remercions plus particulièrement Isabelle Mougenot, pour avoir pris la peine de participer à l'ensemble de nos restitutions et pour nous avoir consacrer de nombreuses et précieuses heures pour travailler sur notre soutenance.

Nous remercions la société Geomatys, et particulièrement sont directeur, Vincent Heurteaux, pour l'intérêt insatiable envers notre travail et les opportunités de formations qui nous ont été offertes.

Enfin nous remercions Gérard Breitel pour son aide généreuse dans notre recherche de logement pour ce stage.

Analyse des techniques de stockage et d'interrogation du web sémantique

Elèves : EV2 J.CHAPEAU et EV2 A.DROZ-BARTHOLET, EN 10

Type de rapport : Rapport de Projet de Fin d'Etudes (PFE)

Référence : PFESIM-EN10-1

Groupe d'acceuil : IRD Montpellier

Tuteur de projet : Jean-Christophe DESCONNETS & Christelle PIERKOT

Soutenance : Lundi 3 décembre 2012 à 13h25

RESUME :
Le web sémantique cherche à rassembler les ressources d'origines différentes. Les méta-données se développent ainsi afin de faciliter la circulation de l'information.
MDWeb est un logiciel de catalogage qui utilise les méta-données afin de référencer les données géographiques. Cet outil utilise une base de données relationnelle. Le but de ce projet est d'explorer la possibilité d'utiliser une base de données assurant le stockage des métadonnées en s'appuyant sur le métamodèle RDF. La première partie du projet consiste en un état de l'art des différentes base de données offrant des possibilités de stockage d'un tel modèle à triplet. Après la mise en place des différentes bases de données, il s'agit de tester les performances de tels systèmes. Ce stage s'inscrit dans une logique de développement orientée vers le web sémantique. Ainsi les travaux effectués durant ce projet seront utilisées pour continuer d'explorer les techniques de stockage et d'interrogation du web sémantique.

ABSTRACT :

The semantic web aim to put many Internet ressources together. Metadatas are expanding to ease information traffic.
MDWeb is a software which gathered data, it use metadata in order to use different sources. This tool use a relationnal database. The goal of this project is to see the possibility of using a database which can store meta-datas by respecting the RDF langage. The first part of the project focuses on a state of the art of the different existing no-SQL solutions. Then, after setting the different database, many tests should be run in order to test the performances of such sytems. This period go with the willing of MDWeb to move toward the semantic Web. Finally this work will be used for further experimentations on storing and queryin technics which are linked to the semantic Web.

Mots-clés :

Table des matières

Glossaire

Fiche de méta-données Une fiche de métadonnées s'appuie sur un profil et correspond à l'ensemble des valeurs qui renseignent les éléments de métadonnées de ce profil.

Meta-données Une méta-données est une données servant à décrire, définir et qualifier une autre donnée.

Namespace (Espace de noms) En XML un namespace sert à identifier l'origine des balises. Cela permet d'utiliser différents vocabulaire au sein d'un même fichier XML.

Norme Une norme est un référentiel standardisé visant à harmoniser une activité donnée.

Ontologie Utilisée notamment pour le web sémantique, une ontologie est un réseau sémantique reliant un ensemble de concept afin de décrire totalement un domaine spécifique.

RDF (Resource Description Framework) modèle organisé sous forme de triplet (sujet, prédicat, objet) permettant de décrire formellement les ressources web et les méta-données afin de faciliter leur traitement automatique.

SPARQL (SPARQL Protocol and RDF Query Language) est un langage de requête pour le RDF recommandé par le W3C (Consortium World Wide Web).

SQL (Structured Query Language) développé par IBM en 1974, ce langage de requête est presque universellement reconnu par les systèmes de gestion de bases de données relationnelles. Il s'agit du langage le plus utilisé pour les bases de données relationnelles.

Thésaurus Liste organisée et hiérarchisée de termes dans un domaine défini. L'ensemble des termes sont liés par des relations de hiérarchie , de synonymie et d'association.

Timestamp Terme anglais signifiant horodatage. Il s'agit d'associer à un élément la date et l'heure des événements qui sont survenus. Cet horodatage réponds en général aux normes définissant l'affichage du temps universel en informatique.

VectorClock est un algorithme permettant de gérer les conflits de versions dans la modification de ressources. Il permet de retracer l'historique des modifications apportées afin de déterminer avec certitude la bonne version.

XML-RDF Syntaxe définie par le W3C pour écrire des fichiers RDF sous la forme de fichiers XML.

Introduction

L'information spatiale est aujourd'hui au cœur de notre monde. Les quantités de ressources géospatiales circulant chaque jour augmentent de manière exponentielle. De nombreuses institutions liées au domaine environnemental utilisent aujourd'hui l'information géographique comme support et aide à la décision. Elles possèdent de nombreux jeux de données spatiaux qu'elles se partagent généralement entre plusieurs groupes d'utilisateurs distincts (e.g.écologue, hydrologue, etc...). Ces ressources géospatiales sont décrites par des méta-données. Ces dernières ont pour effet d'améliorer l'efficacité des recherches d'informations. Afin de pouvoir utiliser des ressources d'origines différentes, il est essentiel de mettre en place des normes pour régir l'utilisation de méta-données.

L'outil au centre du projet est le logiciel MDWeb. Ce dernier est un outil Web de catalogage et de localisation de ressources. Il est diffusé sous Licence LGPL V3. Son développement est assuré par une collaboration entre UMR ESPACE DEV et la société Geomatys . Une de ses principales fonctions est d'être un moteur de recherche de ressources géospatiales. Pour cela, il s'appuie sur l'interrogation des métadonnées géographiques (e.g. emprise spatiale) et thématiques décrivant le contenu et la nature de la donnée (e.g. titre, résumé, mots clés, ...) Il utilise un thésaurus pour faciliter l'indexation sémantique des données et leur interrogation via des interfaces de recherche. L'architecture de cet outil s'articule autour de composants logiciels génériques, basés sur les spécifications de l'OGC. Les normes de l'ISO TC/211 [ISO11] sont utilisées pour structurer les métadonnées qui sont stockées dans une base de données relationnelle. L'un des objectifs du projet MDWeb est de réunir à échelle nationale voire internationale des ressources spatiales d'origines variées.

Aujourd'hui, de nouvelles problématiques liées au partage de ressources entre différentes communautés émergent. La mutualisation des ressources impose une gestion de l'hétérogénéité des normes spécifiques à chaque communauté. Par ailleurs, afin de proposer un service efficace, il est nécessaire de réfléchir à de nouvelles solutions pour améliorer les performances du moteur de recherche, lors du requêtage qu'il soit thématique et/ou spatial.

Une des pistes envisagée est de s' appuyer sur le modèle RDF pour la structuration des méta-données. Ce modèle a été développé par le W3C et propose une organisation sous forme de triplet (sujet, prédicat, objet) qui permet de décrire les ressources et leurs méta-données. Pour stocker les méta-données, au format RDF, plusieurs solutions sont envisageables. Tout d'abord un stockage dans des bases de données non-relationnelles est possible. Ce sont des bases de données qui ont un modèle beaucoup plus flexibles que les bases SQL traditionnelles. Une autre solution est l'utilisation d'un Triple Store qui a pour vocation de stocker des triplets RDF. La dernière solution consiste à conserver l'architecture relationnelle tout en y ajoutant une couche qui transforme ce format en triplets RDF : c'est l'utilisation d'un fichier de mapping.

Le stage a pour objectif de tester les solutions proposées afin de répondre aux problématiques de MDWeb . La première étape consiste à étudier les différentes technologies qui correspondent aux solutions proposées. Puis, après s'être familiarisé avec les outils sélectionnés, il est nécessaire de mettre en place une plate-forme de test afin d'observer les performances des différentes solutions techniques. Enfin une étude des résultats obtenus doit permettre de déterminer les solutions qui permettent de répondre aux différentes problématiques.

Au final ce rapport présente le fruit de ce travail. Dans un premier temps un état de l'art sur les technologies associées aux Web sémantique est présenté. Puis la deuxième partie s'intéresse à la mise en place des différentes solutions trouvées et des tests associés. Enfin la dernière partie

1. www.geomatys.fr/
2. www.opengeospatial.org/
3. www.mdweb-project.org/

s'intéresse aux résultats, conclusions, et perspectives associées à ces différents tests.

Chapitre 1

Contexte, enjeux et méthodologie

Sommaire

1.1 Méta-données

1.1.1 Généralités

Une méta-donnée permet de définir ou décrire une autre donnée. Le terme de méta-données est apparu dans les années 1990[SC310], dans le cadre de la description des ressources Internet. Celles-ci sont, dans le cadre du Web sémantique, des données signifiantes qui permettent de faciliter l'accès au contenu d'une ressource informatique. Par exemple, l'en-tête des documents HTML contient des méta-données. Au minimum cinq éléments, répartis autour de trois domaines, permettent d'identifier et de décrire les ressources documentaires :

Contenu : titre, sujet, description, source, langue, relation, couverture.

Propriété intellectuelle : créateur, éditeur, contributeur, droits d'auteur.

Matérialisation : date, type, format, identifiant.

Dans l'optique du web sémantique, les ressources numériques transportent alors leurs propres méta-données lorsqu'elles sont échangées. Ceci s'applique à tous les types de ressources numériques (texte, son, image, multimédia). Les méta-données sont ainsi l'un des principaux éléments de l'étiquetage des ressources disponibles sur le web[Pol09].

Le potentiel des méta-données est beaucoup plus important que l'utilisation actuelle qui en ai faite par le web 2.0, car elles peuvent faire inter-opérer les ressources informatiques, si elles respectent une norme commune, dans des dictionnaires de données (ou registres de méta-données). On peut alors faire communiquer les bases de données classiques (utilisées dans les progiciels de gestion intégrés) et les données non structurées (documents, images, manipulés en gestion des connaissances. . .).

Afin que les méta-données des ressources d'origines différentes, puissent être utilisées ensemble, il est nécessaire qu'elles soient basées sur le même modèle. En effet l'accroissement du nombre de données disponibles, et d'autant plus dans un domaine des données géospatiales, a rendu nécessaire la création de normes régissant les méta-données. Cette normalisation rend possible le partage cohérent des données via les méta-données [Pie08]. Le Dublin Core est un schéma de méta-données qui permet de décrire des ressources numériques ou physiques et d'établir des relations avec d'autres ressources. Il est composé de 15 éléments de méta-donnée (titre, créateur, éditeur), intellectuels (sujet, description, langue, ...) et relatifs à la propriété intellectuelle. Le Dublin Core fait l'objet de la norme internationale ISO 15836. Il est employé par l'Organisation mondiale de la santé, ainsi que d'autres organisations intergouvernementales.

Dans le cas de MDWeb, bien que la norme Dublin Core soit supporté, c'est une norme plus spécifique aux méta-données géographiques qui a été retenue : la norme ISO 19115. Les normes de méta-données sont des normes qui décrivent les données sur les données, employées pour la structuration des ressources informatiques en général (pas seulement les documents électroniques) et l'interopérabilité informatique. Étant donné les multiples utilisations des méta-données, à la fois dans les ressources informatiques et les systèmes, il est nécessaire d'employer des normes.

1.1.2 La norme ISO 19115 pour les données spatiales

La norme ISO 19115 a le statut de norme internationale depuis 2003. Cette norme abstraite de contenu définit en les organisant par classes toutes les informations que l'on peut mettre à disposition pour décrire la donnée géographique [ISO03]. Cette norme s'est affirmée comme une référence pour l'information géographique. Outre le fait qu'elle est enfin disponible après une longue gestation, ses atouts principaux pour la communauté résident dans son caractère modulaire et extensible qui la rend aisément adaptable.

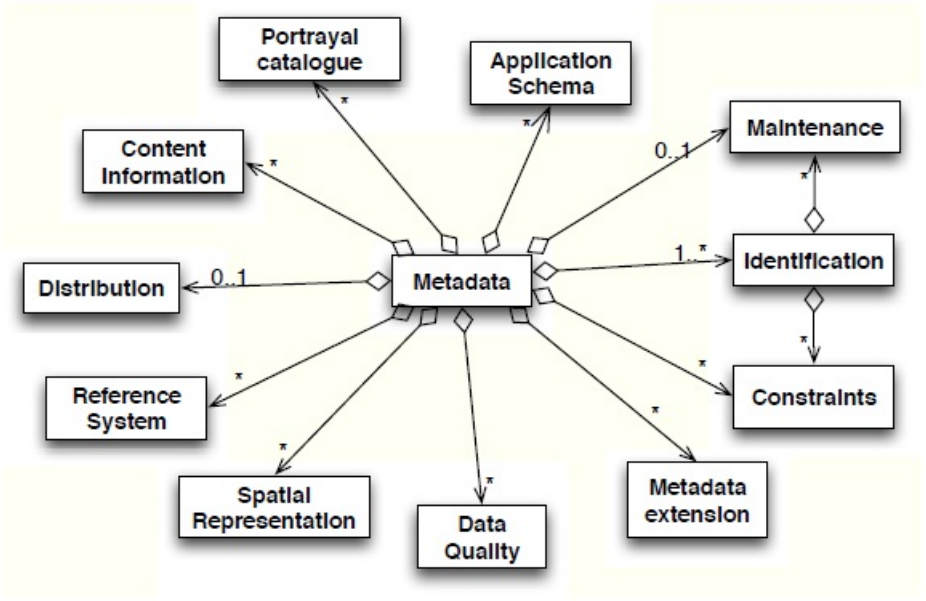

FIGURE 1.1 – Les différentes rubriques de la norme ISO 19115 (formalisme UML)

Au sens de la norme ISO 19115, les éléments de méta-données sont les suivants :
- Informations d'identification : intitulé, description, dates de référence, version, résumé, intervenants, ...
- L'étendu des données
- Des aperçus sur les données
- Informations sur les emplois possibles
- Contraintes légales et de sécurité
- La description du contenu
- Le système de coordonnées
- Les informations de géolocalisation et d'organisation des données

- Des informations sur la qualité des données
- Des mesures de qualité (précision géométrique, temporelle et sémantique, exhaustivité, cohérence logique)
- Des informations de généalogie (Description des sources, description des processus appliqués aux sources)
- Les modalités d'affichage (légendes)
- Les modalités de diffusion
- Les modalités de maintenance

La figure 1.1 [JCD08] présente le méta-modèle de la norme ISO 19115. Ce dernier reprend les éléments cités précédemment

La norme ISO 19115 est complétée par la norme ISO 19115-2 qui définit des méta-données pour l'imagerie, et par ISO 19119 qui décrit les méta-données de service. La spécification technique ISO 19139 définit quand à elle une implémentation XML standardisée pour ISO 19115 [ISO07]. Il y a de nombreux avantages dans la normalisations. Par exemple les moteurs de recherche dédiés à la lecture et au décryptage sémantique de ces données permettent une optimisation et une efficacité accrue des recherches d'information opérées par un internaute ou un ordinateur sur le Web. La valeur ajoutée de cette solution technique repose sur un mode de requête qui écarte les informations parasites. Dans le cadre du Web sémantique c'est un moyen de mettre en place ce système par une optimisation des méthodes et moyens appliqués à la recherche d'information et de documentation dans un système d'information donné :

- Ne rendre visibles et lisibles que les informations pertinentes pour l'utilisateur (avec indice de pertinence).
- Diminuer les risques de désorientation liés à un déluge d'informations (nombre de réponses non pertinentes rapportées à la question posée qui peuvent faire dériver l'internaute) comme c'est le cas aujourd'hui sur internet.
- Par rapport au bruit généré par les recherches plein texte, les méta-données insérées dans les ressources informatiques permettent d'améliorer les recherches d'information sur le Web, comme avec les logiciels utilisés par les bibliothèques. La normalisation permet alors de rendre compatibles de nombreuses données puisque leurs méta-données sont éditées selon le même modèle.

1.2 MDWeb

1.2.1 Présentation de l'outil

Le projet MDWEB a débuté en 2003. Ce logiciel est né d'un besoin de partage d'information . En effet le réseau d'observatoires de Suivi Ecologique à Long Terme (ROSELT) étudie les mécanismes de dégradation des terres. Pour cela il est indispensable de partager les informations relatives à cette problématique sur chaque territoire afin de réaliser des comparaisons [dSedS04].Il a donc fallu organiser les données existantes. De plus les différents laboratoires doivent être capable de modifier ces données ou d'en ajouter de nouvelle. Ainsi MDWEb, à l'origine, a été créer pour répondre à une problématique de mutualisations des ressources géospatiale , afin de soutenir des travaux réalisés en réseaux [JCD08]. La première édition assurait quatre fonctions : référencer et rechercher les ressources pour les localiser et y accéder. Les infrastructures de données spatiales (IDS) répondent à l'évolution rapide d'internet et aux initiatives de standardisation en matière d'exploitation de l'information géographique. Ainsi des organismes internationaux se sont mis en place tels que l'Open Geospatial Consortium ou l'organisation internationale de standardisation (International Standard Organisation, ISO). Ainsi, MDWeb est essentielle pour les applications environementale qui souhaite diffuser, et partager les ressources géospatiales. La structuration doit suivre des normes afin de répondre à la problématique d'interopérabilité. MDWEB propose deux grandes fonctionnalités. Elles s'articulent autour de la gestion des métadonnées ou catalogage et la recherche des ressources afin de localiser, choisir et accéder aux ressources souhaitées et la fonction de localisation [JCD08] Tout d'abord le côté utilisateur de données. MDWeb doit permettre de rechercher des données en interrogeant les métadonnées. Une interrogation se compose de cinq critères :

- le contenu de la ressource
- son type

- son extension spatiale
- son extension temporelle
- le propriétaire de la ressource

A l'issue de cette recherche l'utilisateur peut accéder directement à la ressource qui l'intéresse via un protocole web (HTTP, FTP, etc.) [IE10]. Quant au côté producteur de données, l'outil permet de décrire les nouvelles données en s'appuyant sur les méta-données. Afin d'organiser cette nouvelle ressource un catalogage permet de décrire de manière formel la ressource. Enfin une annotation sémantique permet de rajouter une couche sémantique à cette nouvelle données.

1.2.2 Architecture de MDWeb

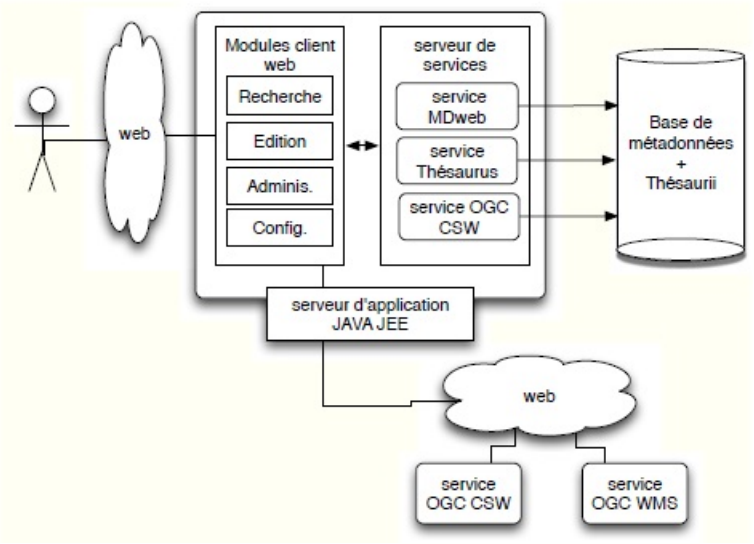

FIGURE 1.2 – Schéma de l'architecture générale de MDWEB (formalisme UML) [JCD08]

La figure 1.2.2 [JCD08] reprend l'architecture du projet MDWeb.Via un navigateur web, il est possible d'accéder aux modules client. C'est à cet endroit qu'on retrouve les fonctionnalités décrites précédemment. Ces outils sont développées à l'aide du Framework JAVA Server Faces.Voici les services qu'offre MDWeb :

- Un service web standardisé de catalogage CSW (Catalog Services for the Web) de l'OGC qui fournit les interfaces de recherche et d'accès aux fiches de méta-données stockées par MDWEB.
- Un service web spécifique à l'outil MDWEB qui fournit les interfaces pour l'édition des méta-données, l'administration et à la configuration de l'outil. – Un service web d'accès aux thésaurus permettant l'exploitation des concepts et des relations stockées dans ces thésaurus.
- Une base de données relationnelle (PostGreSQL) assurant la persistance des méta-données, des thésaurus et des modèles de méta-données utilisés (normes et profils).

1.3 Open Linked Data et RDF

1.3.1 Open Linked Data

Le modèle de l'Open Linked Data rassemble deux modèles :

Linked Data : Ce sont les données sur le web liées à d'autres sources mais qui ne sont pas nécessairement ouvertes aux développeurs. Sauf exception, elles présentent un format structuré comme le format RDF.

Open Data : Données sur le web qui ne sont pas nécessairement liées mais qui sont ouvertes aux développeurs.Les principaux problèmes de l'exploitation des données ouvertes sont de l'ordre technique car les données en masse ne peuvent pas être traitées humainement. Le concept de Web des données appliqué aux données ouvertes met en œuvre 3 mécanismes :

- permettre l'existence de la donnée sur le réseau à travers une URI unique.
- utiliser un langage structuré afin de pouvoir rassembler de nombreuses ressources comme le RDF ou les microdonnées dans le HTML5 (d'où l'importance de normes universelles, appliquées par tous).
- améliorer la qualité de la donnée pour éviter qu'un traitement de mise à disposition ne puisse les altérer.

Un entrepôt de données même avec des erreurs est préférable qu'un entrepôt biaisé. Ainsi, des mécanisme pour la fréquence et l'automatisation des mises à jour de la donnée par les producteurs des données est possible avec un service SPARQL sur ces données. Les données ouvertes ne sont contrôlables que par leurs producteurs (contrôle des mises à jours) et réellement exploitables par d'autres qu'à la condition d'utiliser ces 3 mécanismes.

Le format Open Linked Data est idéal pour le partage d'information.Il regroupe les deux concepts précédents. C'est Tim Berners-Lee qui a défini les quatre piliers pour soutenir le concept de l'Open Linked Data :

- utiliser des adresses URI unique pour identifier les informations
- utiliser des adresses URI HTTP qui existent sur le Web.
- fournir à travers l'URI des renseignements lisibles par les humains et par les machines. En utilisant le mécanisme de redirection HTTP et la variable User-Agent contenue dans les entêtes des requêtes HTTP, un serveur peut afficher une page en XML/RDF pour une machine ou une page HTML pour le navigateur d'une personne ;
- Une Erreur HTTP 404 indique que l'URI utilisé est peu fiable et ne doit pas être réutilisé pour décrire d'autres données.
- Fournir un URI aux ressources externes .

1.3.2 RDF

RDF (Ressource Description Framework) est un langage standardisé par le W3C, il permet de modéliser tout type d'information et de donnée identifiée par une URI (Universal Ressource Identifier, en général accessible sur internet via le protocole HTTP). RDF est un format de fichier destiné à stocker des données afin d'être traitées par des applications et non directement par l'Homme.

Une assertion simple peu être représentée de la manière suivante :

« http ://www.pfe2012.fr/Montpellier a pour créateur Jean Pierre».

Cet exemple a pour sujet http ://www.pfe2012.fr/Montpellier et a une propriété nommée "créateur" qui a pour valeur "Jean Pierre". C'est à ce niveau qu'intervient RDF. Ce dernier permet une représentation simplifiée de cette méta-donnée. Chaque ressource est composée d'un triplet :

Le sujet : ce dont on parle (http ://www.pfe2012.fr/Montpellier)

Le prédicat : une propriété du sujet (le créateur)

L'objet : la valeur de cette propriété (Jean Pierre)

Cette représentation par URI permet d'identifier toute sorte de ressource (physique ou abstraite) de manière unique. Les objets peuvent être des liens vers d'autres ressources (URIs) ou des valeurs littérales (les types doivent être définis en utilisant la syntaxe XML). Cette première solution permet de créer des liens entre plusieurs dépôts de données et évite la redondance de l'information. Admettons qu'une page web possède un créateur et une date de création, le créateur peut être scindé en un nom et un prénom, une date définie par son jour, son mois et son année. Cependant, ces dernières propriétés ne s'appliquent pas qu'à un site web. Il est donc plus cohérent de pointer vers une autre ressource de type personne ou date comme le montre l'exemple présenté par la figure 1.3.

RDF peut être encodé suivant la syntaxe XML, il bénéficie donc des avantages de XML (format standard reconnu) et fonctionne de la même manière. RDF définit donc un langage de balisage spécifique appelé RDF/XML. Du fait de sa structure, RDF permet de naviguer à partir d'une donnée vers une nouvelle ressource : une source mise en relation avec d'autres informations en utilisant un navigateur Web sémantique. Les liens RDF peuvent être utilisés par des moteurs de recherche spécifiques qui vont fournir des recherches plus complexes et permettre des requêtes sur l'ensemble des données. Les résultats des requêtes étant structurés et liés à des pages HTML, ils

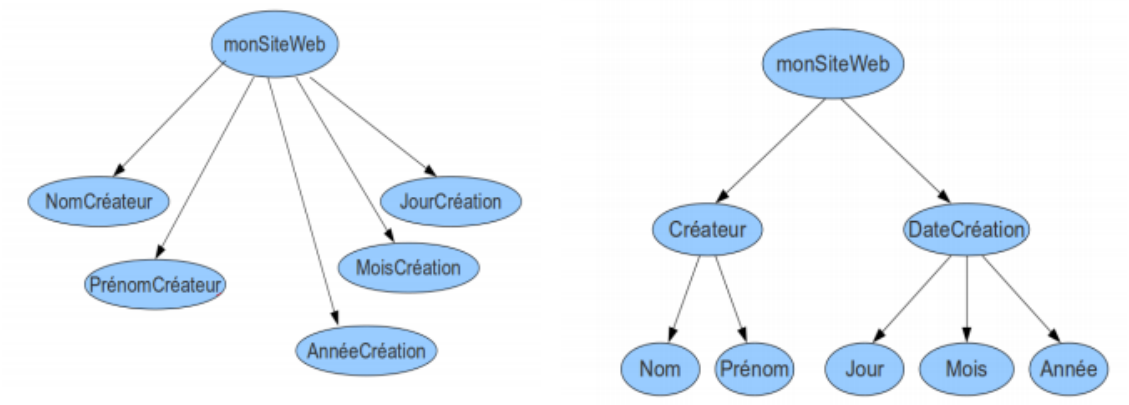

FIGURE 1.3 – Exemple RDF avec ressources

peuvent être réutilisés par d'autres applications. RDF permet de rendre les données accessibles à toutes les applications à travers le Web.

1.4 Méthodologie

1.4.1 Objectifs

MDWeb est un outil déjà développé et distribué, le coup engendré par le changement de base de données, car il nécessite de développer à nouveau bon nombres de composants du logiciel. De plus le nombre de solutions disponibles sur le marché des bases de données est important. La première étape consiste donc en un état de l'art détaillé des nouvelles solutions de bases de données. Cet état de l'art devra aboutir à un comparatif détaillé afin de sélectionner celles qui pourraient être intéressantes. Pour établir ce comparatif, il est indispensable de définir au préalable les critères de choix de manière précise.

1.4.2 Critères pour le choix des bases de données

MDWeb est développé en Java, pour des raisons d'intégration les outils développés dans ce langage seront à privilégier. Une tolérance pourra être accordée aux logiciels n'étant pas développés en Java mais offrant une interface Java complète et distribuée officiellement par l'éditeur. De plus afin de faciliter l'intégration les outils disposant d'une documentation importante seront favorisés, qu'il s'agisse de la documentation officielle fournie par l'éditeur du logiciel tout autant que de la documentation réalisée par la communauté (tutoriaux ou cours). De même une forte communauté sera un gage de possible soutien au développement mais également de pérennité de l'outil. Tout comme la pérennité, la maturité du projet est un caractère essentiel. D'une part elle garantit le sérieux de l'équipe de développement. Mais surtout elle permet de rassurer sur la stabilité de la technologie. De plus une technologie jeune subie en règle générale et de manière régulière des modifications qui peuvent parfois nécessiter de lourdes maintenances pour les applications les utilisant.

MDWeb est également un outil libre, à ce titre, il est inconcevable de faire appel à des bases de données sous licence payante. Les logiciels distribués en Open Source (licence libre dont la modification et la redistribution est autorisée) ainsi que les outils nécessitant le moins de modifications possibles du code existant seront à privilégier.

Un des derniers critères de choix est l'implémentation de la technologie RDF dans l'outil. En effet l'objectif est de pouvoir intégrer la richesse proposée par RDF pour améliorer l'interrogation des méta-données les requêtes. Outre le fait de gérer le RDF, la base de donnée devra également être capable d'interpréter et d'exécuter des requêtes SPARQL. Une vigilance particulière devra être apportée à la version de SPARQL supportée, de grandes différences existent entre chacune d'elles.

La liste suivante récapitule les principaux critères qui devront conditionner et orienter l'état de l'art des bases de données :

Développé en Java Un outil entièrement développé en Java sera préférable (un outil offrant une bonne interface pourra être accepté)

Licence L'outil doit être sous licence libre

SPARQL La base de donnée devra pouvoir être interrogée avec le langage SPARQL. Les dernières versions de SPARQL seront à privilégier.

Documentation La documentation devra être bien fournie tant celle fournie officiellement que celle de la communauté.

Taille de la communauté Une communauté forte et active sera privilégiée.

Maturité du projet Les logiciels disponibles en version de test ou encore instables devront être évités.

1.4.3 Mesure de la performance des technologies

Aux caractéristiques définies précédemment s'ajoutent également des contraintes de performances. En effet, il n'est pas envisageable de perde en robustesse ou en vitesse de réponse des requêtes. Il est donc nécessaire de soumettre les technologies retenues à une série de tests s'approchant au maximum des conditions d'utilisations réelles de l'outil MDWeb.

Parser XML vers RDF

Afin de pouvoir réaliser les tests, il faut tout d'abord réunir un jeu de données. Afin de pouvoir comparer les résultats des technologies retenues à celles de MDWeb, il est nécessaire d'extraire ces données directement du catalogue existant. MDWeb n'offre cependant pas la possibilité d'exporter les fiches en RDF, il faut dont implémenter un outil permettant d'exporter les méta-données au format RDF ou de les transformer du format XML ISO 19139 vers le format RDF/XML. Pour des raisons de rapidité, la deuxième option a été retenue.

Générateur de fiches

Afin de tester les capacités des différentes technologies retenues dans des conditions proches des conditions réelles (pour une implémentation dans MDWeb), nous avons choisi de réaliser des tests en augmentant graduellement la volumétrie des métadonnées. Cela permettra de percevoir les éventuels problèmes de performance liées à la montée en charge de la technologie testée.

Afin de mesurer à la fois la justesse des résultats retournés et le temps de réponse, nous avons choisi de construire notre jeu de données sur une centaine de fiches réelles. Pour réaliser la montée graduelle en volumétrie, nous avons complété ce jeu de métadonnée par des fiches factices. L'analyse des résultats sera ainsi faite sur la base d'un jeu de requêtes pour lequel nous connaissons les résultats retournés. C'est pourquoi pour ces tests seront utilisé un ensemble de 320 fiches tirées de MDWeb, la montée en charge sera ensuite réalisée avec des fiches factices. Ces fiches seront construites de manière à ne pas être des résultats des jeux de requêtes tests (absences des mots clés, bounding box différentes). Ainsi en plus du test de performance, il sera possible de vérifier que l'augmentation du nombre de fiches n'engendre pas d'autres problèmes.

Jeu de requêtes

Dans MDWeb, plusieurs types de requêtes sont utilisées, les requêtes lexicales portant sur la comparaison de chaînes de caractères et des requêtes géospatiales. Il est également possible de combiner ces deux types de requêtes. Les requêtes devront donc tester l'ensemble de ces aspects.

Pour répondre à ces critères plusieurs requêtes ont été exécutées sur MDWeb, celle ci ont ensuites été traduites en SPARQL afin d'être exécutée sur les technologies à tester. La liste suivante présente les 7 requêtes qui seront utilisées pour les tests (les requêtes SPARQL sont disponibles en annexe) :

- Requête lexicale sur les champs titre, résumés et mots clés.
- Requête lexicale sur les champs titre, résumés et mots clés et typage.
- Requête lexicale sur les champs titre, résumés et mots clés avec opérateur logique.
- Requête géospatiale.

- Requête lexicale sur les champs titre, résumés et mots clés et empreinte géospatiale.
- Requête lexicale sur les champs titre, résumés et mots clés et typage avec empreinte géospatiale.
- Requête lexicale sur les champs titre, résumés et mots clés avec opérateur logique et empreinte géospatiale.

Protocole de test

Afin de réaliser des mesures précises, un outil spécialisé sera utilisé : JMeter. Cet outil permet d'envoyer successivement à la base de données 10 fois chaque requête et de récupérer le temps de réponse. Cette opération sera ensuite répétée plusieurs fois avec un nombre croissant de fiches. Les paliers retenus sont : 320, 600, 1200, 2500, 5000, 10 000, 15 000, 25000,50 000 et 100 000 fiches.

Afin de s'approcher au maximum des conditions d'utilisation réelle et pour obtenir un test fiable, le test est réalisé à l'aide de deux machines, l'un hébergeant le système de base de donnée, ainsi qu'un serveur HTTP Post (qu'il a été nécessaire de réaliser) sur lequel JMeter, lancé sur une deuxième machine, envoie les requêtes SPARQL. Il est nécessaire de veiller à ce que l'environnement d'exécution soit identique tout au long des tests (processus en cours d'exécution, vitesse du réseau). Pour cela les deux machines seront sur un réseau isolé, elles seront reliées entre elles par un cable Ethernet RJ45. Aucune application parasite ne devra fonctionner en même temps.

Une fois ces tests réalisés, les mesures réalisées au cours de ces tests , nous permettrons d'évaluer, selon la volumétrie, l'adéquation ou l'inadéquation des différentes technologies. dans un deuxième temps, une analyse détaillée des résultats pourra nous fournir les pistes optimiser le stockage au format RDF ou l'optimisation des requêtes SARQL par l'adjonction d'index. Ou à tout le moins si certaines opérations de développement sont nécessaires avant l'implémentation (développement d'index spécifiques, élaboration de schémas OWL).

Chapitre 2

Etat de l'art des méthodes de stockages NoSQL et RDF

Sommaire

2.1 NoSQL

2.1.1 Généralités

Les bases de données relationnelles sont apparues en 1970 afin d'offrir une alternative au stockage des données par un système de fichiers qui se révèle couteux et peu performant. Ces bases ont connu un rapide engouement tant au niveau universitaire que commercial [VMZ+10]. La force du langage est de se baser sur la logique ensembliste permettant une modélisation relativement simple des bases et facilitant la manipulation des données. Avec ce concept a été introduit la notion de transaction ACID (Atomicity Consistency Isolation Durability). Durant une transaction, plusieurs requêtes sont effectuées, en cas d'erreur, la totalité de la transaction est annulée, permettant au système de revenir dans son état antérieur. De plus ce système de transactions permet de gérer l'accès concurrentiel aux données, chaque transaction ayant la vision de la base de données telle qu'elle était lors de l'initialisation de cette transaction.

Aujourd'hui la donnée est au cœur de l'économie de toutes les entreprises. Si cela parait évident pour des sociétés comme Facebook ou Google où la donnée est la raison même de la société, cela l'est moins pour d'autre. Mais même en industrie la recherche de la performance se fait grâce à l'exploitation des données. Si bien que les systèmes de bases de données relationnelles ne sont pas suffisants pour répondre à toutes les attentes. Les principaux problèmes restent l'augmentation de l'hétérogénéité des données et du volume.

Dès les années 1980 d'autres solutions apparaissent pour pallier à la faiblesse du modèle relationnel face à ces deux problématiques. C'est alors qu'apparait l'informatique décisionnelle et les entrepôts de données. L'objectif est de stocker et journaliser l'ensemble des informations afin de pouvoir les exploiter à des fins d'analyse. Les données sont traitées lors de l'insertion et des agrégats sont définis, on sacrifie donc la notion de non redondance des données pour garantir une grande vitesse de calcul. De plus dans ces entrepôts, la donnée n'est pas modifiable, elle n'est pas volatile. C'est donc pour cela qu'un nouveau système de gestion était nécessaire. L'infographie présentée en figure 2.1 illustre la grande diversité des systèmes de gestion de base de donné. Il est possible d'y distinguer la séparation système de gestion de base de données et entrepôt de données (respectivement operational et analytic). Ainsi que la séparation plus récente relationnel, non relationnel.

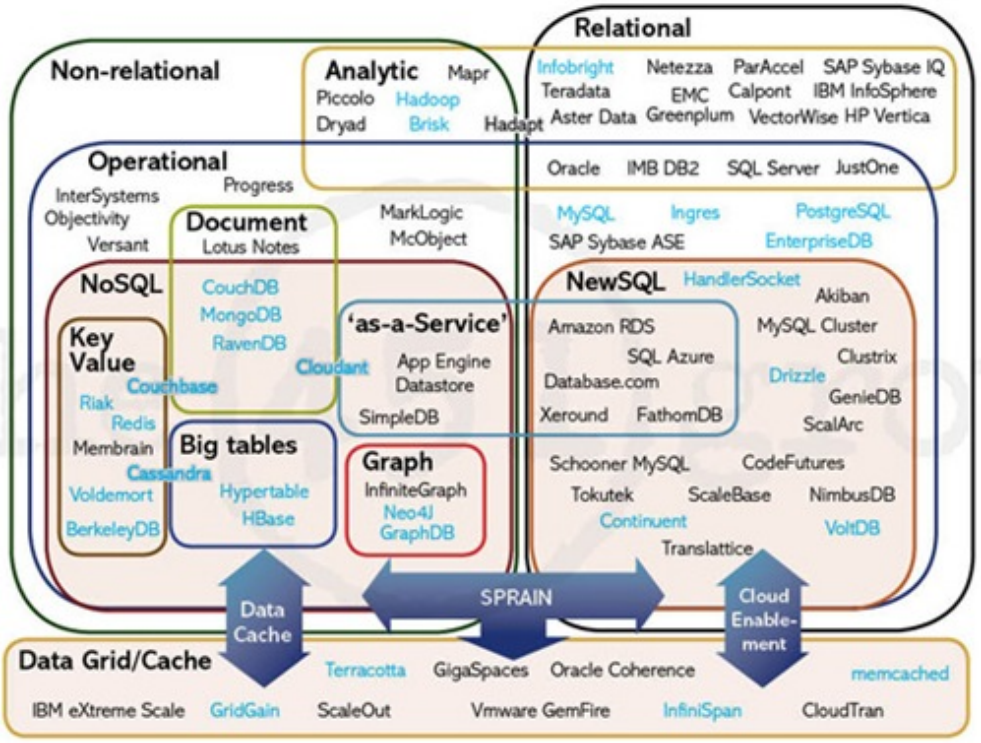

FIGURE 2.1 – Classification des systèmes de gestion de base de données [Sri12]

Si le modèle relationnel s'adapte à la majorité des domaines d'utilisation, l'émergence de nouvelles problématiques telles que le stockage d'objets, le volume toujours croissant des données ou encore l'hétérogénéité des données a régulièrement poussé les développeurs à regarder d'autre modes de stockage. Mais ces alternatives telles que les bases de données objets ou les bases de données XML n'ont pas réussi à s'imposer [SSK11]. Depuis quelques années une nouvelle catégorie de systèmes de gestion de bases de données émerge, le NoSQL. Bien que le concept et le terme soit défini dès 1998, le NoSQL connaitra un regain de popularité et d'intérêt en 2006 avec la parution d'un article de Google, décrivant son système BigTable, un système s'appuyant sur la logique NoSQL [Tiw11, CDG+08].

NoSQL est la combinaison des mots No et SQL, qui est le langage inventé par IBM le plus répandu pour l'interrogation des bases de données relationnelles. La volonté affichée du concept est d'offrir une alternative à l'hégémonie des systèmes relationnels jugés parfois trop complexes et couteux [Lai09]. Cependant les avis divergent et la tendance est aujourd'hui de définir le NoSQL comme "Not Only SQL" [Bur11, Tiw11]. Le concept devient une alternative qui ne vise pas à supplanter le modèle relationnel, mais à venir le compléter dans quelques cas particuliers.

Principalement soutenu par les géants du web, ce nouveau mode de stockage leur permet de gérer un grand nombre de transactions et de maintenir une latence faible et viable sur une quantité de données astronomique. A titre d'exemple, Jay Parikh, vive-président responsable des infrastructures informatiques de Facebook, déclarait lors d'une conférence de presse le 22 août 2012 que

l'infrastructure de Facebook permettait de stocker chaque jour 500To de données et était capable de traiter 105To d'informations en 30 minutes [Tam12]. Ces quantités de données imposent une nouvelle architecture distribuée et le NoSQL a été développé afin de répondre à ce besoin en facilitant l'ajout et la suppression de serveurs. Pour permettre cette facilité d'extension des bases, le NoSQL sacrifie le caractère ACID du modèle relationnel, au profit d'un caractère appelé BASE (Basicaly Available, Soft-State, Enventualy consistency) [VMZ+10, Bur11]. Les bases NoSQL mettent ainsi en avant la disponibilité de la ressource et la possibilité de distribuer les données au détriment de leur cohérence.

Si les bases NoSQL doivent sacrifier la cohérence, c'est en réponse au théorème présenté par Eric Brewer en 2000, appelé théorème CAP (Consistency, Availability, Partition tolerance) et largement adopté aujourd'hui [Bre00]. Chaque modèle de base de données, ne peut assumer que deux des trois propriétés que sont :

La cohérence (Consistency) : L'ensemble des éléments du système distribué voient et contiennent la même donnée au même moment.

La disponibilité (Availability) : Toute requête doit recevoir une réponse.

La résistance au morcellement (Partition Tolerance) : Chaque sous ensemble du système peut travailler de manière indépendante et autonome en cas de perte de connexion entre les sous-ensembles.

Les bases de données relationnelles se portent vers le respect de la cohérence et de la disponibilité, alors que le NoSQL tente de répondre au besoin de cohérence/résistance au morcellement et disponibilité/résistance au morcellement comme illustré par la figure 2.2.

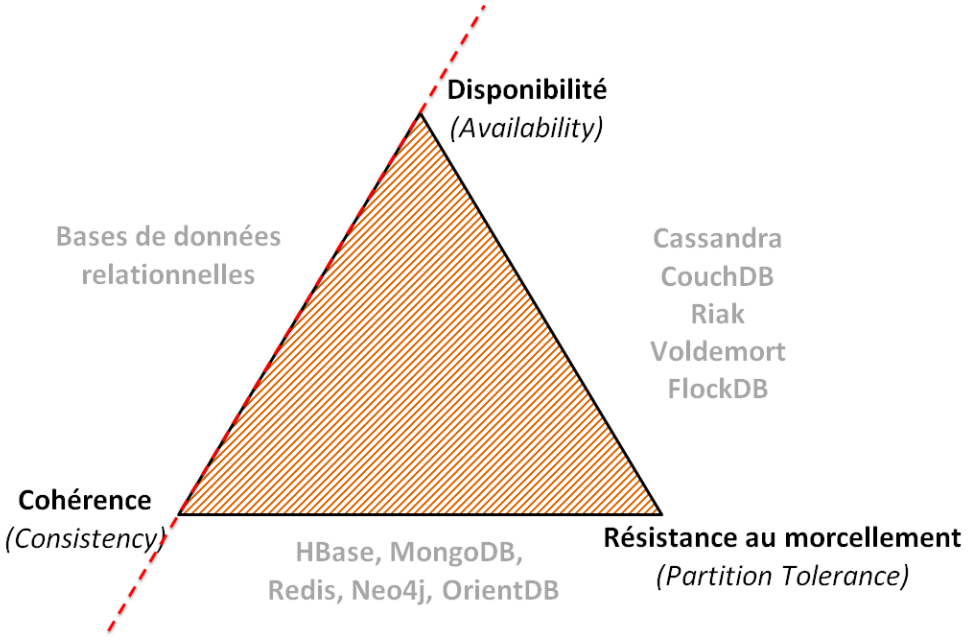

FIGURE 2.2 – Illustration du théorème CAP

Les différentes solutions NoSQL se partagent donc en fonction de leur orientation. Celles respectant le caractère cohérent des données utilisent souvent un système de transaction proche de celui des bases de données relationnelles comme par exemple Neo4j [Tea12]. Alors que d'autres, plus orientées disponibilité, règlent les conflits d'accès multiples après enregistrement avec un système de gestion de versions tel que dans CouchDB [ALS09]. Plusieurs méthodes sont utilisées pour gérer les conflits, telles que le timestamp ou le vectorclock [SSK11].

2.1.2 Typologie

Le NoSQL est un concept englobant plusieurs modèles de base de données, c'est une famille de produits [Tiw11]. Il existe quatre types distincts de base de données NoSQL : [SSK11, Bur11, Tiw11, Bre00]

Le modèle clé/valeur : Il s'agit du modèle le plus simple, chaque objet est identifié par une clé unique.

Le modèle orienté document : Les éléments sont des documents, souvent au format JSON ou XML. Chaque document peut être composé de plusieurs couples champ/valeur.

Le modèle orienté graph : Dans ce modèle, il est fait usage de nœuds et de relations auxquels il est généralement possible d'associer des propriétés.

Le modèle orienté colonnes : Cette structure est à la fois la plus proche des bases de données relationnelles et la plus complexe à appréhender. Les données sont stockées sous forme de tables, mais le nombre de colonnes est variable, ce qui évite notamment le stockage de valeur NULL.

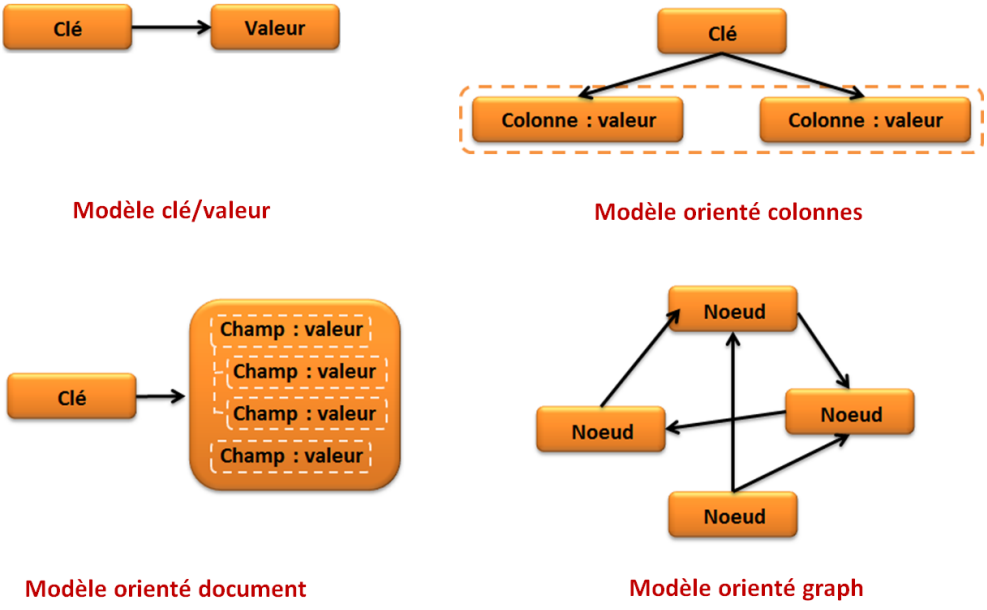

FIGURE 2.3 – Les quatre types de bases NoSQL

La figure 2.3 reprend et illustre cette classification en 4 catégories.

2.2 Les différentes catégories du NoSQL

2.2.1 Modèle clé/valeurs

Présentation

Le modèle de clé/valeur est le modèle le plus simple. Chaque objet est identifié par une clé unique. Suivant les logiciels de gestion de base de données utilisés, les objets peuvent être des entiers, des chaînes de caractère, des objets sérialisés ou même des documents (PDF, images ou tout autre forme numérique). Les opérations sur la base se limitent à 4 opérations (CRUD) : Create (Création d'un enregistrement), Read (Récupération de la valeur pour une clé donnée), Update (Modification de la valeur pour une clé donnée), Delete (Suppression du couple clé/valeur) [Fou11]. Une base de données clé/valeur est donc relativement similaire à une table de hachage persistante [Ipp09]. Cependant la force de ces bases de données est de permettre le stockage distribué sur plusieurs machines, et donc de gérer la réplication, ce que ne ferait pas une table de hachage.

L'une des solutions pionnières dans le domaine des bases clé/valeur a été la base Dynamo, développée par Amazon [DHJ+07]. Le système permet d'effectuer des opérations que sur une paire clé/valeur à la fois, il n'est donc pas possible de créer des références entre les valeurs [SSK11]. N'étant pas distribuée gratuitement, cette solution a principalement été une source d'inspiration pour d'autres solutions. Dynamo s'oriente vers les propriétés Availability et Partion Tolerance du théorème CAP [Tiw11, DHJ+07]. Cela permet à la base d'offrir une très grande disponibilité. En édition, une réponse positive est envoyée quand au bout d'un certain temps un nombre suffisant

de serveur ont répondu (paramètre fixé par l'utilisateur appelé quorum). Les conflits sont ensuite gérés par un système de vectorclock [SSK11].

Avantages et domaines d'utilisation

La force des bases de données à clé/valeur réside dans leur très grande simplicité. En offrant un jeu de requêtes très restreint et une architecture épurée, la base de données gagne en disponibilité [Ipp09].

Ces bases sont utilisées pour des données fréquemment utilisées et souvent temporaires. Il existe des bases de données qui fonctionnent uniquement sur la RAM de la machine et utilisées uniquement pour le cache telles que EHCache et Memcached [Tiw11]. Cependant elles ne peuvent être utilisées que pour des données avec très peu de relations [Fou11].

Le principal défaut de ces bases est que la quasi-totalité des fonctions que l'utilisateur attend d'un système de gestion de base de données doivent être gérées côté utilisateur. Les requête ne se faisant que sur une paire clé/valeur, les opérateurs tel que SELECT doivent être implémentés par le développeur. Les logiciels n'offrant en général aucun moyen de gérer les liens entre les données, tout ce qui se rapproche des jointures en relationnel doit être implanté dans le programme utilisant la base [SSK11]. De plus, pour les conflits en écriture, la résolution est laissée à l'utilisateur. La base propose en général un système similaire à un système de gestion de versions classique (tel que Subversion). L'utilisateur doit donc déterminer sa politique de résolution en fonction de ses besoins (conserver la version la plus récente, récupérer les éléments modifiés des deux versions . . .) [Fiq10a].

Solutions disponibles

Un grand nombre de bases à système de clé/valeur est disponible sur le marché, telles que : Oracle Berkeley DB, Dynamo, Tokyo Cabinet, Redis, Memcached, MemcacheDB, Scalaris, CouchBase, Riak, Voldemort. Et cette liste n'est pas exhaustive. Par la suite deux bases de données seront détaillées : Voldemort et Riak. Elles ont été retenues car elles sont Open Source et utilisées par des sociétés d'envergures.

Voldemort

Voldemort est un outil développé et utilisé par LinkedIn (réseau social de professionnels). La société LinkedIn a ensuite publié Voldemort sous licence Apache 2 (OpenSource). La base est simplifiée à l'extrême et n'offre que 3 opérations : get, put et delete [Lin12]. Le concept utilisé est illustré par le méta schéma présenté en figure 2.4.

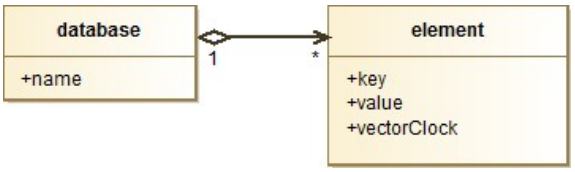

FIGURE 2.4 – Méta-Schéma de la base de donnée Voldemort

Tout comme Dynamo, Voldemort est orienté disponibilité et tolérance au morcellement [Kre09]. La cohérence des données étant là aussi gérée par un système de gestion de versions via un vectorclock. Cette base de donnée est développée en Java et les requêtes peuvent se faire directement via le langage Java ou via une API HTTP Rest.

Riak

Riak est un outil développé par Basho et est utilisé notamment par Mozilla et GitHub. Directement inspiré par Dynamo, Riak est distribué sous licence Appache 2. Riak offre un peu plus de fonctionnalités que Voldemort, notamment la possibilité de gérer des liens entre les données qui peuvent ensuite être parcourus comme un graphe [Fiq10a]. De plus Riak offre la possibilité de regrouper les enregistrements dans des "buckets" qui peuvent être assimilés à des tables. Cette structure est représentée par le méta schéma en figure 2.5.

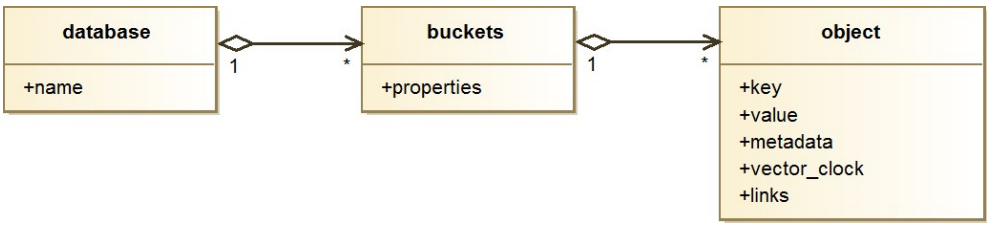

FIGURE 2.5 – Méta-Schéma de la base de donnée Riak

Avec Riak, la valeur peut être un champ classique (chaîne de caractères, entier, flottant) mais également un document numérique. Tout comme Dynamo et Voldemort, Riak est orienté disponibilité et tolérance au morcellement en offrant un système de gestion des conflits [Bas12]. Même si il n'est pas développé en Java, Riak offre une implantation Java ainsi qu'une API HTTP Rest.

2.2.2 Modèle orienté colonnes

Présentation

Le modèle orienté colonnes est celui qui en apparence ressemble le plus au modèle relationnel. On y retrouve les notions de tables, de colonnes et de lignes. Cependant alors qu'en relationnel le seul élément dynamique est la ligne, les colonnes étant définies de manière définitive par le modèle de la base, les bases de données orientées colonnes permettent à certaines lignes d'avoir des colonnes, alors qu'une autre n'aura pas la même. Les colonnes étant définies dynamiquement, il n'y a plus d'espace utilisé pour stocker les valeurs NULL [Tiw11]. L'enregistrement n'est plus indexé uniquement via une clé primaire, mais pas plusieurs coordonnées composées d'une clé pour la ligne (similaire à la clé primaire) mais également par la ou les colonnes comme l'illustre la figure 2.6.

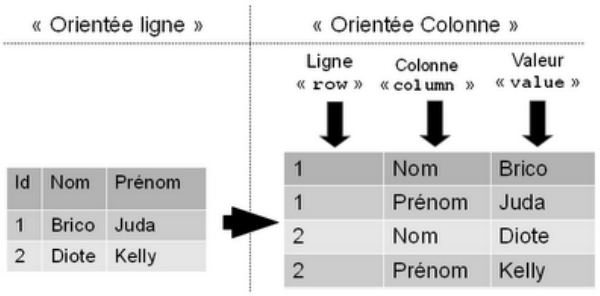

FIGURE 2.6 – Comparaison d'une base relationnelle et orientée colonne [Mor12]

Le plus petit élément de la table n'est donc plus une ligne, mais une cellule. L'accès en lecture et en écriture est donc optimisé car les opérations se font sur des blocs de données plus petit [CST+10]. Dès lors ce système devient très performant pour l'analyse et le traitement de données nombreuses [Fou11]. Finalement le modèle orienté colonne, se trouve à mi-chemin entre le modèle relationnel et le modèle clé/valeur.

Avantages et domaines d'utilisation

Le modèle orienté colonnes offre de nombreux avantages, dont celui d'avoir un schéma très flexible, de nouvelles colonnes peuvent être définies très facilement. De plus les données étant stockées par colonne et non par ligne, il devient plus facile de répartir ces données sur une architecture distribuée [CDG+08]. En effet dans une base relationnelle, la plus petite entité est la ligne (1 dimension pour l'indexation), alors que dans une base orientée colonne, chaque colonne est indexée (plusieurs dimensions pour l'indexation), la plus petite entité est donc une cellule. L'indexation par colonne permet de pouvoir facilement répartir les colonnes sur différents serveurs et le fait d'avoir une entité plus petite augmente la vitesse de calcul. La vitesse d'exécution des requêtes est bien supérieure à celle d'une base de données relationnelle, même si la vitesse de lecture est inférieure à

celle de l'écriture, écriture qui nécessite parfois de réécrire l'ensemble de la colonne pour toute les lignes du à l'indexation de cette colonne.

Les bases orientées colonnes bien que relativement similaires aux bases de données relationnelles, n'offrent pas la possibilité de gérer facilement et efficacement les relations entre enregistrements. De même les requêtes restent assez minimalistes. Cependant à la différence de beaucoup d'autres solutions NoSQL elles offrent la possibilité d'effectuer des requêtes sur une sélection de colonnes (Range Query) telles que « toutes les colonnes comprises entre x et y ». Ce modèle est adapté pour un fort volume de données qui sont très peu interconnectées et qui sont peu modifiées. Google l'utilise notamment pour l'indexage des sites pour son moteur de recherche, et Facebook pour les profils des utilisateurs.

Solutions disponibles

HBase

HBase est un système de gestion de base de données développé par la fondation Apache et distribué sous licence Apache 2. Depuis 2010, ce système est notamment utilisé par Facebook en remplacement de Cassandra. Hbase fonctionne avec Hadoop [Whi10] et bénéficie du soutien de grandes entreprises telles que Yahoo ! [Fiq09]. HBase est souvent qualifié de clone de BigTable, car il reprend l'ensemble des grands concepts définis par Google, comme l'illustre le méta-schéma en figure 2.7.

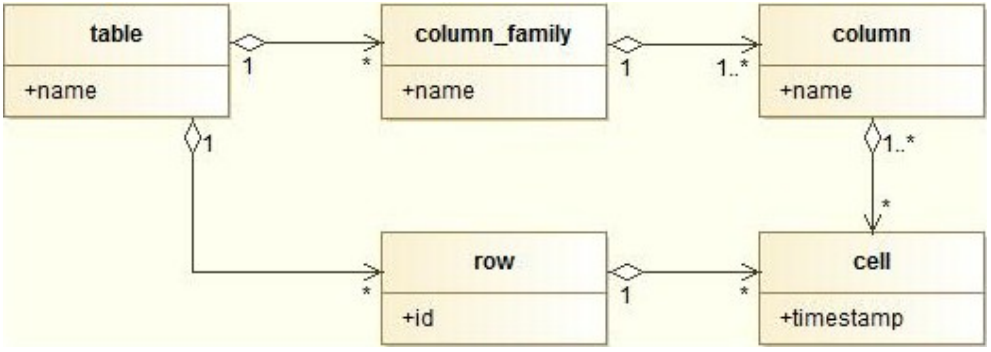

FIGURE 2.7 – Méta-Schéma de la base de donnée HBase

HBase fait le choix d'une base aux données cohérentes plus que disponibles [Apa12b]. Développée en Java elle propose également une interface HTTP Rest pour ses requêtes.

Cassandra

Cassandra est avec HBase, un des deux principaux projets de base de données orientée colonnes de la fondation Apache. Développé à l'origine par Facebook, Cassandra a été cédé à la fondation Apache qui le distribue maintenant en OpenSource. Cassandra s'inspire non seulement de BigTable, mais aussi de Dynamo, une base de données orientée clé/valeur, ce qui place ce système à mi-chemin entre les deux catégories. De ce fait, le méta-schéma de Cassandra est légèrement différent de celui des autres bases orientées colonnes (voir figure 2.8) [Cos12].

Ce choix a été fait de manière à optimiser au maximum la vitesse d'écriture. De ce fait, Cassandra ne renvoie jamais d'erreur en écriture. Toutes les modifications sont stockées dans un buffer local puis enregistrées sur le disque lorsque le buffer est plein ou sur demande [LM09]. Cassandra est donc résolument orienté vers la disponibilité du système et principalement pour l'écriture [Apa12a].

2.2.3 Modèle orienté documents

Présentation

Ce type de base de données est destiné à stocker des documents en offrant une grande souplesse. Ces documents sont un ensemble de couple champ/valeur. Les documents ne sont pas figé et peuvent recevoir de nouveaux attributs à n'importe quel moment. Ces documents généralement au format JSON ou XML offrent une très grande souplesse de stockage des informations.

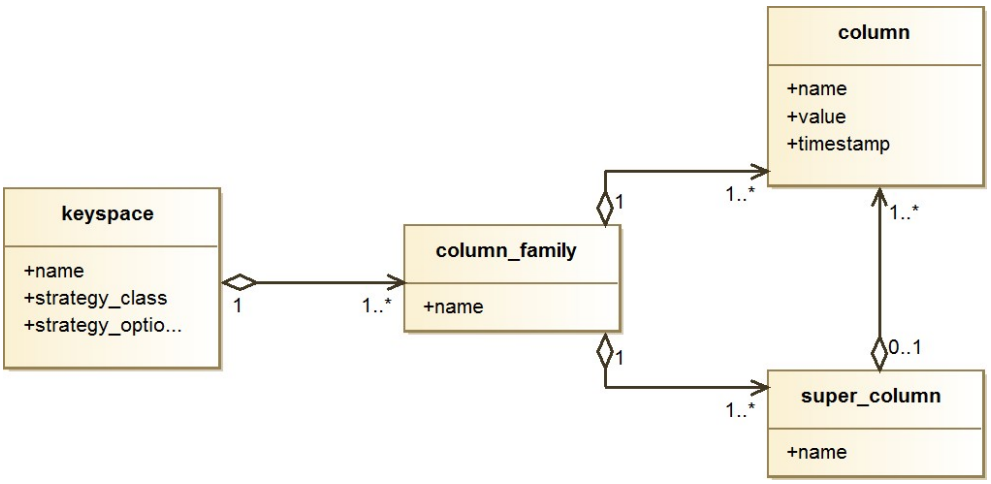

FIGURE 2.8 – Méta-Schéma de la base de donnée Cassandra

En outre, le contenu d'un document ne se limite pas à des attributs simples, il peut également contenir des tableaux, voir inclure un autre document. Par exemple, il est possible de stocker les commentaires liés à un article directement dans le document concernant l'article, sous forme d'un tableau d'objets dépendants. Bien que cette catégorie soit sûrement la plus mature des bases de données relationnelles [Fou11], sa caractérisation reste difficile, car c'est la famille de bases NoSQL la plus diversifiée [Fiq10c].

Avantages et domaines d'utilisation

Les bases de données orientées document s'avèrent particulièrement utiles dans les cas suivants : [Per12]

- Persistance de profils utilisateurs, de regroupement de contenus destinés à une même page, de données de sessions.
- Cache d'informations sous une forme structurée.
- Stockage de volumes très importants de données pour lesquelles la modélisation relationnelle aurait entraîné une limitation des possibilités de partitionnement et de réplication.

Le stockage hiérarchisé permet également un stockage et un accès relativement simple aux fichiers RDF, XML. La structure hiérarchique autorisant la représentation de relations one-to-one et one-to-many. Elle permet le chargement des documents sans aucun traitement de jointure.

Solutions disponibles

Dans le domaine des bases de données orientées documents, les deux acteurs principaux sont MongoDB et CouchDB. Les deux solutions sont relativement similaires et seront détaillées par la suite.

MongoDB

MongoDB est une base de données orientée documents qui connait un succès croissant depuis quelques temps. Distribuée en OpenSource par la société 10Gen, MongoDB est développée en C++. La base de données utilise un dérivé du JSON pour le transport et le stockage des documents, le BSON (sérialisation binaire d'un JSON). Cet outil a notamment été adopté par New York Times et le site SourceForge pour la gestion de leur base de données. Dans MongoDB, les documents sont stockés dans des collections, qui permettent une meilleur organisation de la base. La figure 2.9 représente le méta-schéma de la base de donnée.

MongoDB offre de nombreuses fonctionnalités intéressantes dans le cadre du stockage de méta-données géographiques [CD10] :

- Possibilité d'écrire des scripts MapReduce en JavaScript afin d'effectuer des traitements sur un grand volume de données.
- Gestion des requêtes géographiques. MongoDB cherche à faire face à PostGIS sur ce créneau, en mettant en avant une plus grande facilité d'utilisation que l'extension de PostgreSQL.

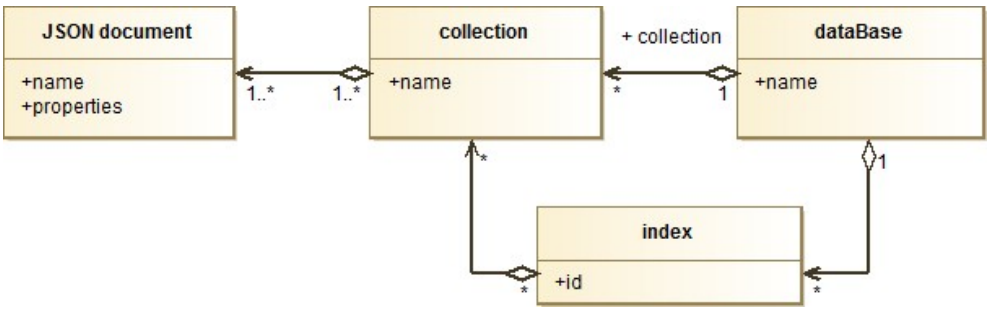

FIGURE 2.9 – Méta-Schéma de la base de donnée MongoDB

– Capacité de réplication et de partitionnement sur plusieurs instances.

CouchDB

Apache CouchDB est un système de gestion de base de données orienté documents, écrit en langage Erlang (langage de programmation utilisé pour les applications temps réel et distribuées) et distribué sous licence Apache. Conçu pour le web, il a été développé pour pouvoir être réparti sur de multiples serveurs. Au lieu d'être ordonnée en lignes et en colonnes, la base de données CouchDB est une collection de documents JSON. Ces JSON peuvent être obtenus facilement via une API HTTP Rest incluse dans le programme. On peut ainsi interroger un serveur CouchDB directement avec un navigateur web ou exécuter des requêtes avec JavaScript.

Tout comme MongoDB, il autorise des opérations de type MapReduce, qui s'avère très efficace lorsque la base de données est importante et répartie. Cependant CouchDB offre également la possibilité d'ajouter des "fichiers joints" aux documents JSON tels que des images, des PDF. Les concepts des deux bases de données restent tout de même relativement similaires comme l'illustre le méta schéma de CouchDB présenté en figure 2.10.

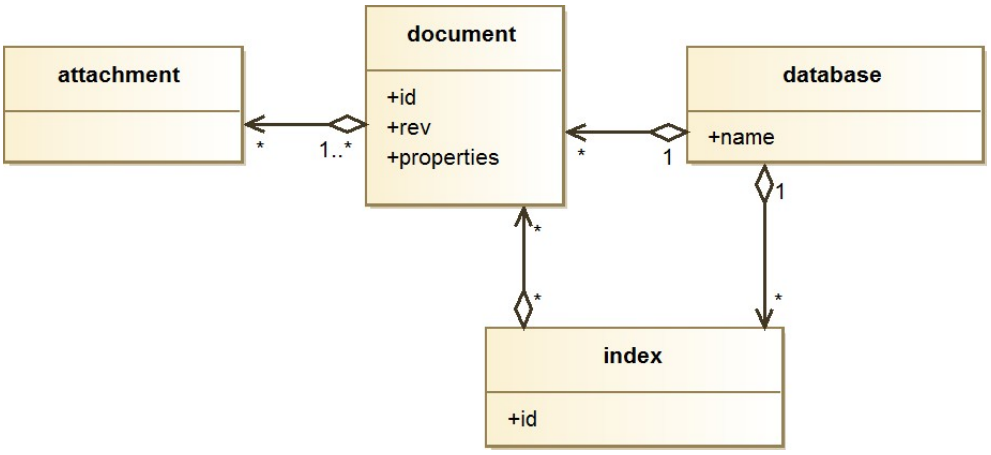

FIGURE 2.10 – Méta-Schéma de la base de donnée CouchDB

Ainsi CouchDB présente une architecture attaché au monde du Web, avec une interface d' échange HTTP, tandis que MongoDB a fait des choix d'un modèle tourné vers la performance et le nombre important de requêtes possibles en une seconde. De plus CouchDB offre un système similaire au système de transaction des bases relationnelles lui permettant de respecter le caractère ACID des bases de données.

2.2.4 Modèle orienté graphe

Présentation

Les bases de données orientées graphe sont une implémentation directe de la théorie des graphes. Ces bases de données sont constituées de nœuds et de relations. Les nœuds sont donc reliés par des

relations, chacune des entités pouvant avoir des propriétés, c'est le modèle du property graph ou graphe à propriété [MT92]. Cette structure en fait la base de donnée de prédilection pour représenter les données organisées en réseaux, tels que les réseaux sociaux. Dans ce cas d'utilisation, les nœuds sont des personnes ou des groupes, et les relations représentent les différentes connexions entre chacun de ces acteurs. Ainsi le géant Twitter utilise FlockDB, une base de données graphes, afin de modéliser son réseau social [Twi12]. Ces graphes peuvent être orientés, les relations ont alors des directions, par exemple dans le cas du graphe social « est dans le groupe », ou non, comme par exemple une relation d'amitié. Enfin certains modèles poussent le niveau d'abstraction plus loin en implémentant la notion d'hyper graph. Or un hypergraphe est un graphe dans lequel les relations ne sont alors plus limitées à deux nœuds, et dans lequel elles peuvent également pointer vers d'autre relations [PM06].

Avantage et domaine d'utilisation

Le modèle graphe est particulièrement adapté dans le cas de données fortement connectées. En permettant de parcourir le graphe de voisin en voisin avec un système de traversal (algorithme de parcours de graphe), la base orientée graphe supprime les jointures multiples, couteuses en temps car nécessitant de parcourir la totalité des index de table. En effet lors d'une jointure sur une table relationnelle il est nécessaire de parcourir plusieurs fois les index de la table (une fois par jointure) alors que pour une base graphe un seul parcours de l'index est nécessaire pour trouver le nœud de départ, la jointure étant remplacer par les relations.

Une autre utilisation, plus particulière au monde des méta-données est la simplicité de représentation du modèle RDF. Ce modèle s'articule autour du concept de triplet (sujet/prédicat/objet). Le graphe permet de représenter facilement ce triplet, le sujet et l'objet seraient représentés par deux nœuds et le prédicat est l'arc entre les deux entités [Té]. De plus il est possible d'appliquer à cette base de donnée tous les algorithmes de la théorie des graphes tel qu'un algorithme de centralité.

En outre les bases de données graphe comme la majorité des bases NoSQL ne demandent pas de schémas prédéfinis au préalable, ce qui leur offre une grande capacité d'adaptabilité.

Solutions disponibles

Comme pour les autres catégories de bases NoSQL il existe de nombreuses solutions telles que FlockDB, HyperGraphDB, Neo4j, OrientDB, Dex, InfiniteGraph. Par la suite, les quatre premières seront examinées plus en détails.

FlockDB

FlockDB est le système de gestion de base de données développé par Twitter. Distribué en OpenSource sous licence Apache 2, cet outil permet de gérer un graphe simple et non un graphe à propriétés [Twi12]. La base ne travail qu'avec des identifiants. Elle est utilisée par Twitter pour le suivit des "Tweets" (messages sur son réseau social) et savoir qui souhaite lire les Tweets de quel utilisateur. La figure 2.11 montre comment la base gère ce système de graphe simple.

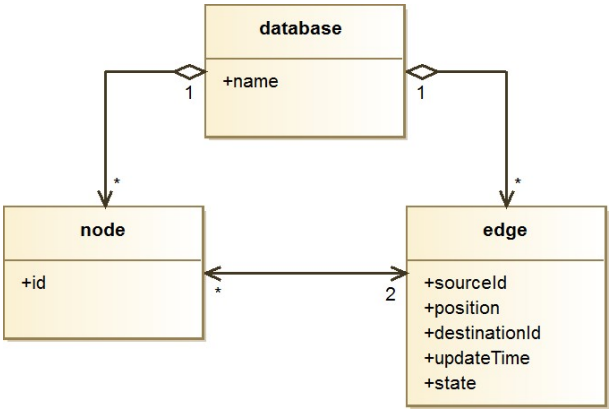

FIGURE 2.11 – Méta-Schéma de la base de donnée FlockDB

FlockDB ne gérant pas réellement de propriétés mais uniquement des relations, elle n'est pas vraiment confrontée aux problèmes de cohérences de données. Une relation ou un nœud existe ou n'existe pas, les problèmes de cohérence sont donc plus simples à gérer, et la base peut donc se tourner vers une haute disponibilité. Cependant FlockDB n'offre pas de possibilité de parcours de graphe, il se limite à l'obtention des premiers voisins d'un nœud. Elle est donc vraiment limitée à la gestion de liens entre personnes.

HyperGraphDB

HyperGraphDB est un outil développé par Kobriw Software distribué en OpenSource dont la particularité est de gérer les hypergraphes. Les noeuds et relations sont toujours présents, mais il devient possible de lier plus de deux noeuds entre eux par une seule relation. Il est également possible de faire pointer une relation vers une autre relation [Ior10]. La figure 2.12 montre le méta schéma d'HyperGraphDB sur lequel il est possible de repérer les différences liées au fait qu'il s'agisse d'un hypergraphe et non d'un graphe simple.

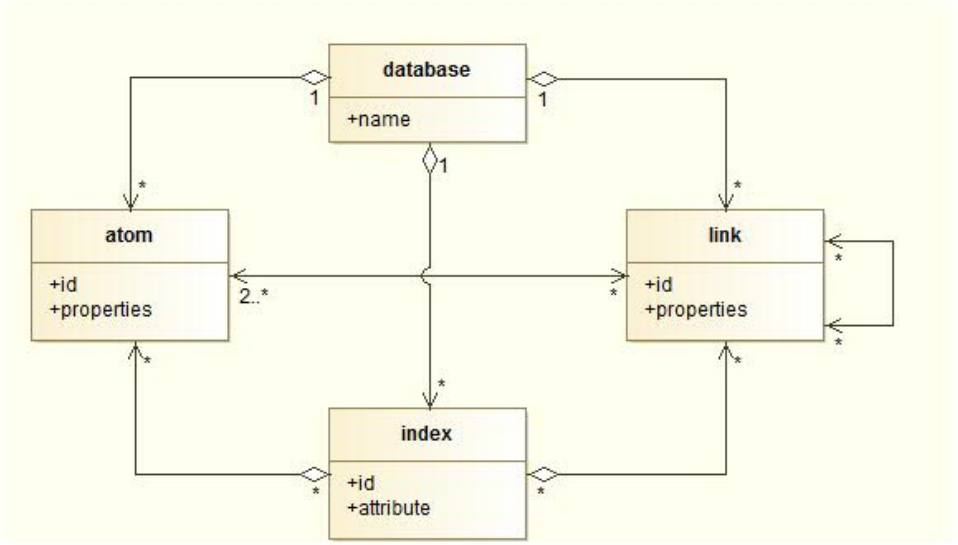

FIGURE 2.12 – Méta-Schéma de la base de donnée HyperGraphDB

HyperGraphDB permet également d'ajouter des propriétés aux nœuds et aux relations. De plus il est également possible de stocker des objets Java sérialisés directement dans la base, HyperGraphDB étant développé en Java. La base HyperGraphDB permet également la gestion des RDF et peut être utilisée comme base ontologique [Ior10].

Neo4j

Neo4j est sûrement l'outil le plus visible sur la scène des bases de données orientées graphe. Distribué par la société NeoTechnology et entièrement développé en Java, on lui reproche cependant d'être distribué sous licence GPL3, qui nécessite l'achat d'une licence pour l'utilisation en entreprise. Malgré tout Neo4j reste très populaire et est utilisé par des sociétés telles que Cisco ou Adobe. Neo4j est un exemple d'implantation directe du property graph. Il est également possible d'ajouter des index (gérés via Apache Lucene). La structure est résumée par le méta-schéma présenté en figure 2.13.

Pour le moment Neo4j ne gère pas le multi instances, cependant les développeurs y travaillent [Fiq10b]. A noter que Neo4j gère aisément plusieurs milliards de nœuds sur une machine, ce qui convient déjà à la plus part des projets. En revanche Neo4j offre un système de transaction similaire à celui des bases relationnelles ainsi qu'un large panel de langages pour les requêtes tels que Cypher et SPARQL.

Neo4j propose un composant qui gère le RDF : neo-rdf-sail (import de RDF depuis fichiers XML, gestion des triplets, requêtes spécifiques), mais son développement est aujourd'hui arrêté. Cependant il existe d'autres solutions open Source qui peuvent gérer du RDF dont Tinkerpop

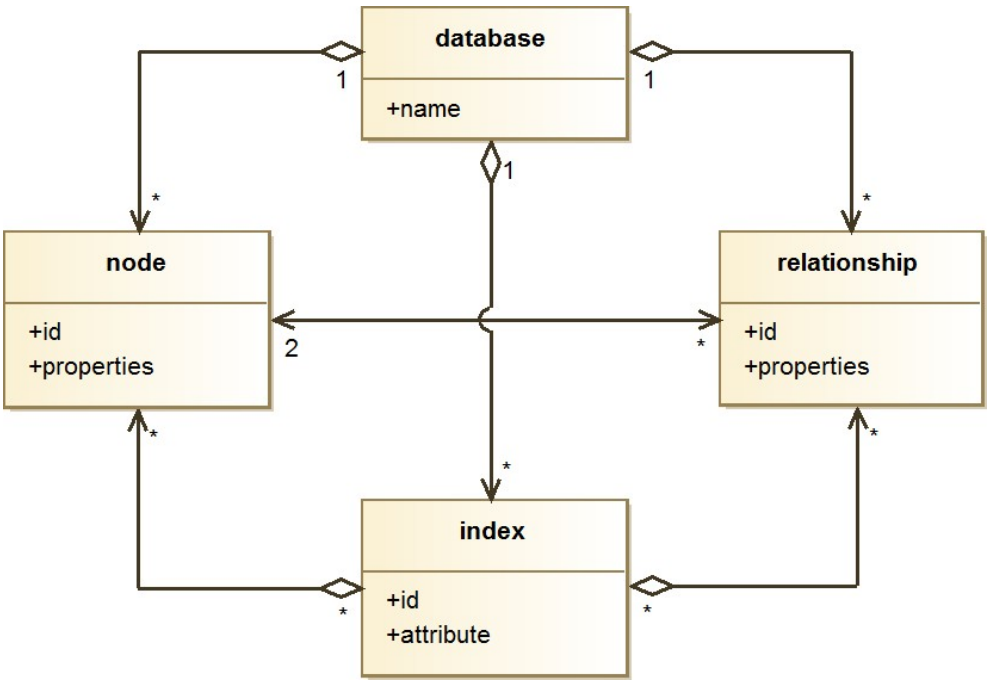

FIGURE 2.13 – Méta-Schéma de la base de donnée Neo4j et OrientGraphDB

[Suv]. Ce projet offre un panel d'outils puissant pour gérer des données RDF, sans être attaché à un modèle de base de données particulier.

OrientDB

Proposé en OpenSource (licence Apache 2) par la société OrientTechnologies, OrientDB est un outil offrant à la fois une base orientée graphe et une base orientée document. Cependant il ne s'agit pas totalement d'une base graphe et documents en même temps, il est nécessaire de choisir le positionnement de la base. Dans le cas d'une base de données graphe, les nœuds et les relations peuvent avoir des propriétés, les relations sont bi-directionnelles et la base de données intègre directement TinkerPop. Dans le cas d'une base de données orientée documents il est toujours possible d'avoir des relations entre les documents mais unidirectionnelles (pour avoir du bidirectionnel il est donc nécessaire de créer plusieurs relations et de les gérer coté client), cependant la base de données est plus rapide et plus légère car les relations ne sont pas des enregistrements mais uniquement des liens entre documents [Gar12]. Le méta schéma d'OrientGraphDB est donc le même que celui de Neo4j (voir figure 2.13).

OrientDB est développé en Java et offre le même panel de langages d'interrogation que Neo4j (Cypher, SPARQL) auquel s'ajoute un dérivé du SQL. OrientDB offre également l'avantage d'avoir un module gérant les données géographiques développé par la société Geomatys et propose un système de transaction respectant les propriétés ACID.

2.3 Les TripleStore

2.3.1 Généralités

Un triplestore est une base de données conçue pour le stockage et la récupération de données RDF (Resource Description Framework). Tout comme une base de données relationnelle, un triplestore stocke des données et il les récupère via un langage de requête. Mais contrairement à une base de données relationnelle, un triplestore ne stocke qu'un seul type de données : le triplet. Elle n'a donc pas besoin de phase d'initialisation pour enregistrer de nouvelles données. C'est-à-dire qu'elle n'a pas besoin de créer des tables comme dans une base de données relationnelle. De plus, un triplestore est optimisé pour le stockage d'un grand nombre de triplets et pour la récupération de ces triplets à l'aide du langage de requête SPARQL. Certains triplestores peuvent stocker des

milliards de triplets RDF.

Dans ce domaine trois grands noms ressortent de par leur maturité : Jena, AllegroGraph et Sesame. Tout d'abord il est important de noter que Jena et Sesame ne sont pas des triplestores à proprement parler. Nativement ils ne permettent pas de stocker de manière persistante les données, il faut leur adjoindre un module qui se base sur une base relationnelle pour stocker les données. Concernant AllegroGraph, il n'est pas open source : au-dessus de 5 millions de triplets RDF, il est nécessaire d'acheter la licence. Compte tenu de cette limitation il ne sera pas étudié plus en détails.

2.3.2 Comparatif des solutions existantes

Jena

Jena est un projet distribué sous license Apache 2. Ce système comporte des outils Java afin de traiter et enregistrer des données au format RDF. Les données peuvent être exploitées sous trois langages différents : XML, N-triples, format Turtle. Il permet également de définir des ontologies. Les requêtes se font via le langage SPARQL. Enfin ce produit est pourvu de nombreux outils de gestions/exploitations des données RDF. Cette API Java permet tout d'abord l'import/export de données au format RDF au format XML. Outre les outils pour effectuer des requêtes Jena est pourvu d'opérations d'union, d'intersection et de différence [MBD04]. De plus Jena permet d'organiser le contenu de la base de données au format RDF à l'aide de conteneurs. Il en existe 3 types :

- le BAG (collection non-ordonnée)
- l'ALT (collection non-ordonnée)
- le SEQ (collection ordonnée)

Un point important est que Jena n'utilise pas de schéma fixe de base de donnée, ici on utilise les ontologies (OWL Ontology Web Language) [Fra12]. Lors d'un ajout d'instance, il n'est pas nécessaire de stocker l'ontologie dans une base de données. Ainsi il est inutile de créer un schéma avec des tables et des colonnes. L'avantage majeur de ce système est qu'il est possible d'intégrer de nombreux modèles de sources différents en RDF. Grâce au langage OWL il est possible de contrôler la validité des données ou de la classification.

Sesame

Sesame est un outil Open source pour la gestion de données RDF créé par Dutch Softwar dans un projet de création de web sémantique. Tout comme Jena, cette outil gère le modèle du triplet RDF. Tout d'abord ce système supporte le langage de requêtes : SPARQL et SeRQL. Un des outil intéressant de ce Triple Store est l'API Alibaba qui est utilisée pour faire du mapping sur deux ontologies en Java. Il est alors possible d'utiliser des ontologies spécifiques telles que RSS, FOAF ou encore Dublin Core directement depuis Java. Mais Sesame offre également d'autres outils : LuceneSail permet de faire des recherches de textes, IndexingSail permet d'effectuer des recherches géospatiales et/ou de textes. Néanmoins Sesame est plus simple que Jena : pas d'inférence, ni de gestion d'ontologie OWL. Il comporte aussi une interface web qui permet de configurer rapidement un triple store.

2.4 Choix d'une technologie pour MDWeb

2.4.1 Premières conclusions

Le NoSQL fait beaucoup parler de lui, cependant, il est nécessaire de rester prudent sur l'emploi de cette nouvelle génération de systèmes de gestion de bases de données. Le NoSQL est en effet adapté dans des situations précises, mais pour la plupart des emplois, un système de gestion de bases de données relationnelles peu tout à fait convenir. De même il est nécessaire de choisir le bon type de base de données NoSQL. La figure 2.14 présente un classement des différentes catégories de bases de données en fonction du volume de données et de la complexité des données (hétérogénéité et relations).

Dans le cadre de MDWeb et de l'exploitation de méta-données géographiques d'une manière plus large, le volume des données n'est pas réellement un problème, et le modèle relationnel répond tout à fait aux besoins. Cependant aujourd'hui la volonté affichée est de pouvoir stocker ces méta-données au format RDF pour une plus grande souplesse. Mais également de pouvoir gérer de

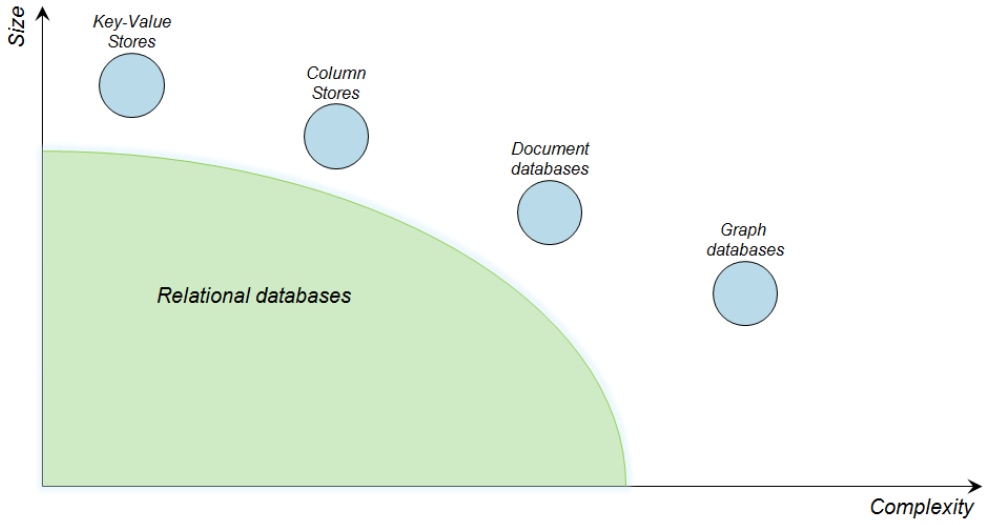

FIGURE 2.14 – Classement des bases NoSQL selon le volume et la complexité des données [Tea12]

manière plus efficace la recherche spatiale ainsi que les liens sémantiques et ontologiques. Les bases de données clé/valeur et orientées colonnes ne semblent donc pas adaptées car elles privilégient le stockage d'une grande quantité de données et ne permettent pas une gestion efficace des relations entre les données.

Parmi les autres solutions présentées précédemment (bases de données orientées documents, bases de données graphes et TripleStore), la plupart offrent la possibilité de gérer le RDF, proposent une gestion sous forme de graphe ou implémentent le langage SPARQL. Du simple point de vue du modèle de la base, plusieurs solutions semblent donc intéressantes à étudier plus en détail. En effet pour l'exploitation du RDF, même si les différents types de bases gèrent le RDF, elles ne le font pas au même niveau de granularité. Comme l'illustre la figure 2.15, le document permet de gérer le fichier RDF en tant que tel, il est possible d'accéder aux triplets, mais l'ensemble des relations, et notamment la représentation sous forme de graphe, ne se fait qu'entre documents. Les TripleStore, spécifiquement dédiés au stockage du RDF offrent la possibilité de gérer au niveau du triplet. La base graphe enfin permet de gérer le RDF au niveau des éléments composant le triplet (sujet, prédicat et attribut).

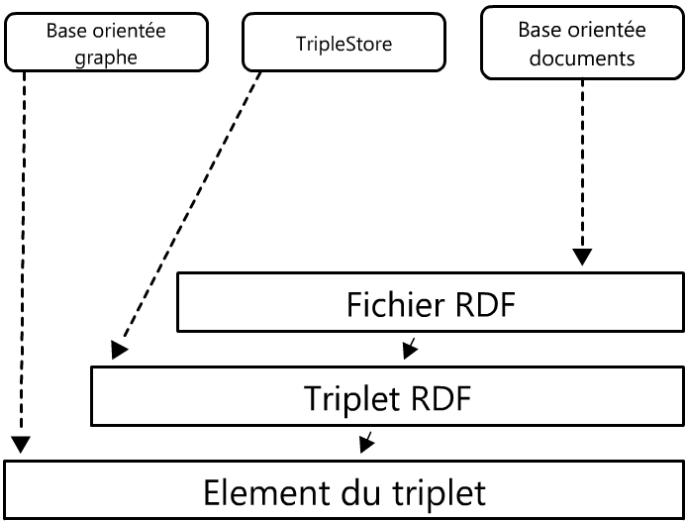

FIGURE 2.15 – Niveau de granularité dans la gestion du RDF pour les bases NoSQL

2.4.2 Avantages de la représentation sous forme de graphe

Les normes régissant la manipulation des données telles que Dublin Core ou ISO 19115 peuvent se modéliser facilement par des graphe de même le RDF ainsi que les thésaurus. Il semble important qu'une base gérant des méta-données offre ces possibilités. Même si une base de données graphe semble alors évidente, il ne faut pas négliger les deux autres solutions (documents et TripleStore) qui proposent elles aussi une gestion sous forme de graphe.

La représentation sous forme de graphe offre de nombreux avantages, le premier est l'exploitation direct de la représentation sous forme de schéma des normes concernant les méta-données comme l'illustre la figure 2.16. Mais surtout elle permet de gérer plusieurs normes en même temps. En effet il existe presque autant de normes qu'il existe de domaines d'études (biologie, géographie, ethnologie ...) et il est difficile de gérer cette hétérogénéité des données. Le graphe sur la figure 2.16 montre comment il est possible de faire correspondre deux normes à l'aide de nœud abstrait. Par exemple le nœud « Titre » représente de manière abstraite ce qu'est un titre, les relations avec les URIs vers chaque norme permettent de définir ce qu'est un titre pour chacune de ces normes. Ce procédé peut être mis en parallèle avec le concept de classe abstraite en programmation objet.

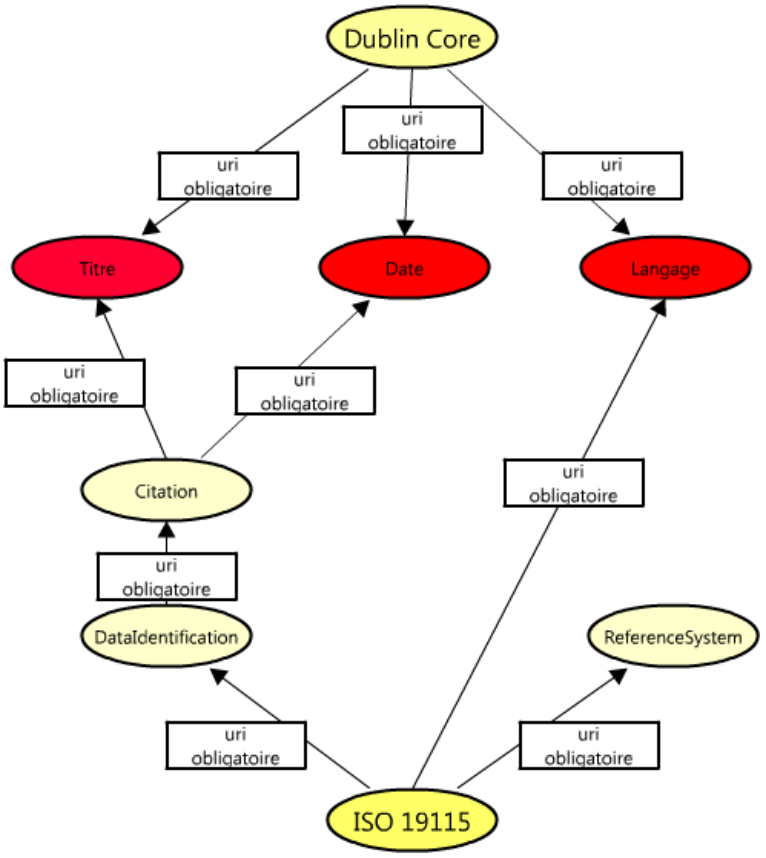

FIGURE 2.16 – Exemple de graphe avec deux extraits de norme

Si il est possible de représenter les normes sous forme de graphe, il est également possible de représenter les fiches (un enregistrement contenant l'ensemble des méta-données relatives à une ressource). La figure 2.17 montre comment une fiche peut être enregistrée sous forme de graphe, où les nœuds contiennent les valeurs et les relations contiennent les prédicats. Cette représentation en elle-même n'apporte que peu de plus-value, le véritable intérêt réside dans le couplage du graphe de la norme avec la fiche de méta-données comme le montre la figure 2.18.

Dans l'exemple de la figure 2.18, la fiche est enregistrée suivant la norme Dublin Core. On voit cependant que grâce aux nœuds abstraits et à la structure en graphe, il est possible de faire une recherche dans un environnement répondant à la norme ISO19115. Cette structure permet de gérer facilement et efficacement des données hétérogènes provenant de plusieurs domaines de recherche et répondant à des normes diverses.

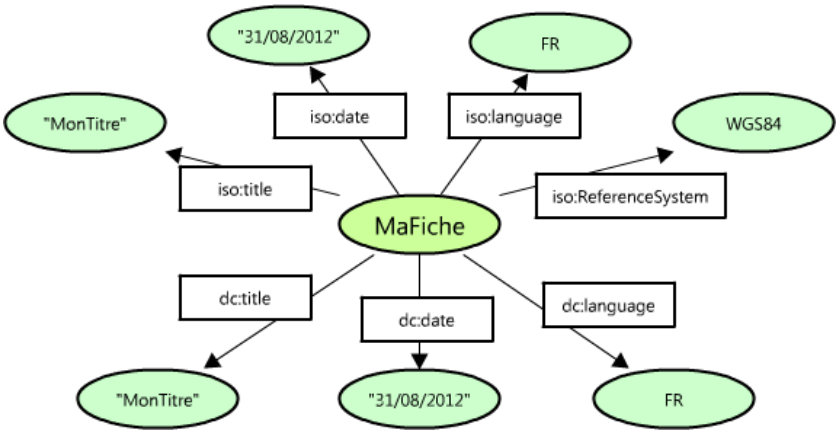

FIGURE 2.17 – Représentation sous forme de graphe d'un fichier RDF

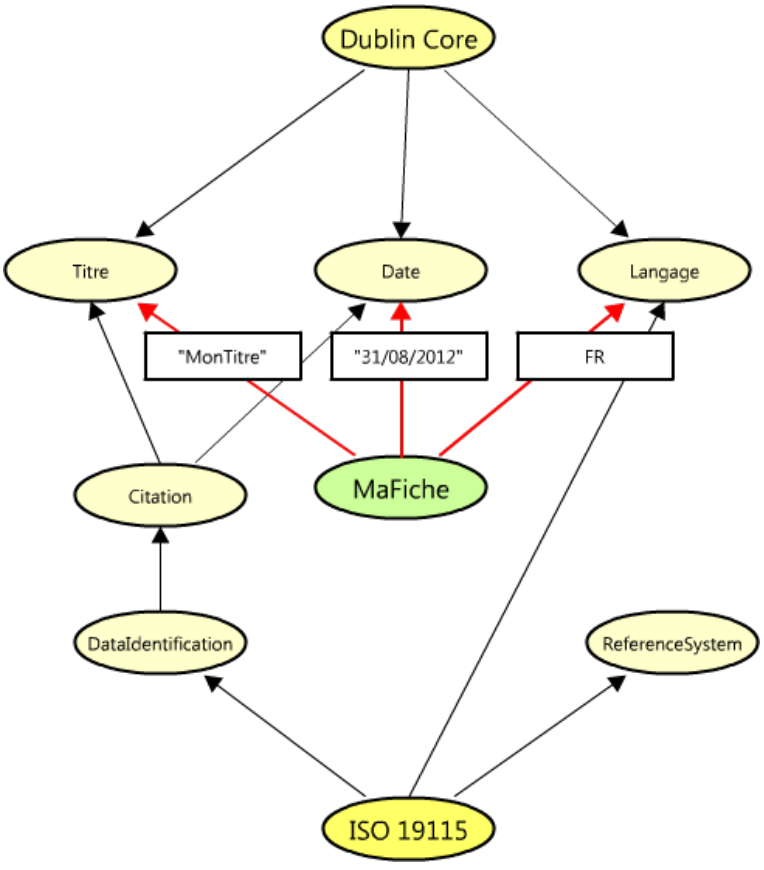

FIGURE 2.18 – Enregistrement d'une fiche dans une structure graphe

Un autre exemple d'utilisation de la structure graphe pour la gestion des méta-données géographiques, est l'élaboration d'un arbre pour indexer les fiches. Dans une base relationnelle classique, pour restreindre la recherche à une zone géographique, l'utilisateur définie une zone à l'aide de 4 coordonnées le minimum et le maximum de latitude et de longitude. Une fois ce cadre défini, lors de la requête le système de gestion de base de données va effectuer des opérations d'intersection et d'inclusion avec l'ensemble des enregistrements. Donc pour une base avec des milliers de fiches, il sera nécessaire de faire autant d'opérations qu'il y a de fiches. Avec un arbre comme illustré par le graphe de gauche de la figure 2.19 où chaque nœud possède également 4 coordonnées définissant son emprise géo-spatiale, il est possible de réduire considérablement le nombre d'opérations. Ainsi avec un arbre tel que celui représenté, quel que soit le nombre de fiches il y aura toujours un

nombre d'opérations constant pour trouver l'ensemble des fiches concernant la France par exemple. En effet lors d'une requête on effectue une opération d'intersection/inclusion pour chaque nœud, si on trouve une correspondance on continue avec les nœuds fils, sinon on ignore la branche. Dans cet exemple, on va effectuer une opération sur Europe et Afrique, puis sur Espagne et France, soit seulement 4 opérations et ce quelques soit le nombre de fiches. Il est ensuite possible d'affiner les recherches en effectuant l'opération sur l'ensemble des résultats obtenus par parcours de l'arbre, par exemple dans le cas où l'utilisateur souhaite une emprise plus fine que celle décrite par l'arbre. De même il est possible des proposer des précisions où des généralisations à d'autres zones géographiques en direct via le parcours de l'arbre.

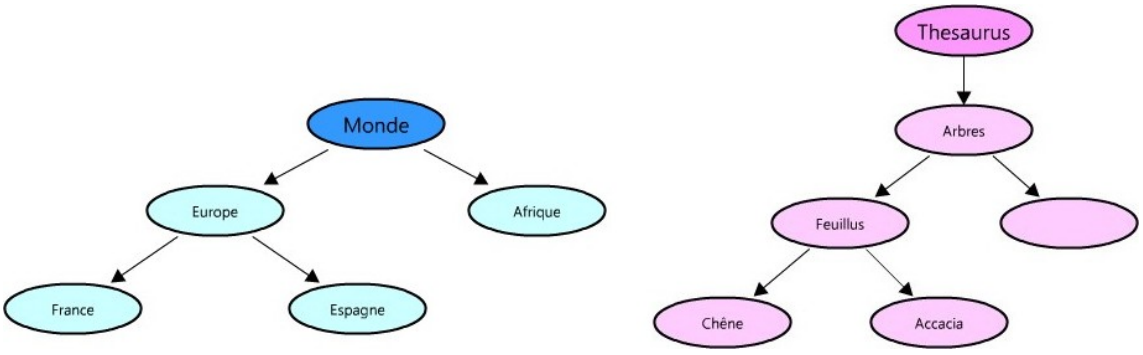

FIGURE 2.19 – Exemple de graphe géographique et sémantique

La structure graphe peut également être utilisée pour les thésaurus. A la recherche classique par mots clés, il est possible de joindre un graphe tel que celui présenté par le graphe à droite de la figure 2.19. Il est alors possible d'effectuer facilement et rapidement des recherches avec des opérations de précision (passer de feuillus à chênes) de généralisation (passer de feuillus à arbres). Mais il devient aussi possible d'utiliser des inférences entre les différents thèmes.

2.4.3 Choix de technologies à tester

Les bases de données graphes, orientées documents et les TripleStores n'apportent pas tout à fait les mêmes avantages. Si une base de données graphe semble offrir la plus grande souplesse, elles ne sont pas non plus dédiées au stockage du RDF comme les TripleStores, elles demeurent donc moins performantes pour des requêtes classiques. Il semble donc intéressant de comparer ces différentes solutions. Le tableau 2.1 présente les différents points d'intérêt ayant guidé le choix de ces solutions.

Par la suite, les solutions suivantes seront implémentées et testées sur une plateforme de test :

- La base de données OrientDB. Cette base de données graphe a été retenue car elle propose également le stockage par document. De plus elle est entièrement développée en Java et est distribuée sous licence Apache 2.
- Les bases de données orientées documents MongoDB et CouchDB. Ces deux distributions ont été retenues car elles sont aujourd'hui les produits le plus aboutis dans le domaine du NoSQL. En plus d'être également distribuée sous licence OpenSource elles permettent de gérer la représentation sous forme de graphe.
- Le TripleStore Jena (grâce à une API permettant la persistance des données) car il est OpenSource et gère les ontologies.

TABLE 2.1 – Tableau récapitulatif partiel des technologies NoSQL

Nom	Type	Licence	Java	Communauté	Orientation CAP
CouchDB	Documents	License Apache 2	Oui	Importante	AP
MongoDB	Documents	GNU Affero GPL	Oui	Importante	CP
Neo4j	Graphe	GNU GPL v3	Oui	Importante	CP
OrientDB	Graphe	License Apache 2	Oui	Moyenne	CP
HyperGraphDB	Graphe	LGPL	Oui	Moyenne	AP
FlockDB	Graphe	License Apache 2	Non	Moyenne	AP
Jena	Triple Store	License Apache 2	Oui	Importante	AP
AllegroGraph	Triple Store	Closed Source	Oui	Moyenne	AC
Sesame	Triple Store	BSD-3-Clause	Oui	Importante	AP

Chapitre 3

Mise en place du banc de test

Sommaire

3.1 Constitution du jeu de données

Les tests seront réalisés sur un jeu de données de 320 fiches. Ces fiches sont issues de MDWeb. Pour compléter ces 320 fiches et réaliser la montée en charge, des fiches factices générées pour l'occasion seront utilisée.

3.1.1 Conversion des méta-données en RDF

Solutions existantes

Le peuplement des bases de données à tester nécessite la conversion des métadonnées XML ISO 19139 vers le format RDF/XML puis leur import dans la base de données. Il existe des outils open-source pour certaines étapes mais pas toutes.Aussi, il a été nécessaire de développer un outil pour cette conversion. Après un rapide tour d'horizon, plusieurs librairies se dégagent :

Analyseur XML : Cette étape consiste en la lecture du fichier XML afin d'en ressortir l'architecture et les données. En Java, deux grandes bibliothèques de distingues : Sax et Dom. Alors que Dom charge en mémoire l'intégralité du graphe XML, Sax gère chaque élément avec un système d'événement [And10].

Transformation au format RDF : Pour cette étape aucune biliothèque n'est à notre connaissance disponible en Java, d'où la nécessité de créer un outil.

Import du RDF dans la base de donnée : Les deux grands outils pour manipuler le RDF sont Jena et Sesame. Tous deux offrent des fonctionnalités relativement semblables surtout du point de vue de l'import de fichiers RDF.

Pour le développement de l'outil, aucun des analyseurs existants n'a été utilisé. Un simple parser texte est utilisé.

Nous décrivons dans le paragraphe qui suit, les classes et méthodes implémentées dans l'analyseur texte qui a été spécifiquement développé.

Description de l'analyseur XML

Le programme se décompose en plusieurs classes (voir Figure 3.1) :

Parser.class La classe Parser réalise la lecture ligne par ligne du fichier XML, à chaque élément (début de document, ouverture/fermeture de baliser, attribut, valeur) le Parser notifie les éléments récupérés à une autre classe, le ParserManager. L'algorithme de la méthode parse() de cette classe est présentée si dessous.

MyParserManager.class Cette classe, qui implémente l'interface ParserManager, réalise la conversion du XML vers le RDF. A chaque ouverture de balise, c'est à dire à chaque incrémentation du niveau d'indentation, la classe va instancier une nouvel objet de type Hierarchy. A chaque fermeture de balise, les triplets nécessaires sont créés et instanciés par l'intermédiaire de la classe Triplet, la classe diminue également le niveau d'indentation de 1. Cette méthode permet de conserver la structure de la fiche (classe, sous classe et instances).

ExportToRDF.class Cette classe permet d'exporter les triplets obtenus (et passés en paramètre au constructeur sous la forme d'une liste de "Triplet") dans un fichier RDF. Ce fichier est formaté sous la forme XML-RDF.

Hierarchy.class Cette classe sert à instancier les niveaux d'indentation dans le fichier XML.

Triplet.class Cette classe permet d'instancier les triplets obtenus.

Ce fonctionnement est repris par le diagramme d'état représenté en Figure 3.2.

Algorithme 1: Algoritme de la méthode parse()

```
Résultat : Document converti en RDF
initialization;
tant que il y a des lignes dans le fichier XML faire
    tant que il reste des éléments dans la ligne faire
        si on trouve une balise ouvrante alors
            on notifie de manager;
        fin
        si on trouve une balise fermante alors
            on notifie de manager;
        fin
        si on trouve un attribut alors
            on notifie de manager;
        fin
        si on trouve une valeur litteral alors
            on notifie de manager;
        fin
    fin
fin
```

Une description plus approfondie est disponible dans la Javadoc fournie avec le code source dans l'archive numérique.

Résultat et perspectives

Dans sa version actuelle, le parser réalisé permet de transformer toute fiche de méta-données au format XML ISO19139 [ISO07] en une fiche au format RDF. Ces fiches sont au format XML-RDF et sont valides lorsqu'elles sont testées avec le validateur du W3C (http ://www.w3c.org/RDF/Validator/). Cette transformation se fait dans un temps raisonnable puisqu'il est possible de transformer plus de 100 fiches par seconde (sur un ordinateur portable grand public). De plus ces fiches sont correctes, c'est à dire qu'il n'y a pas de pertes d'informations entre la fiche au format XML et celle au format RDF.

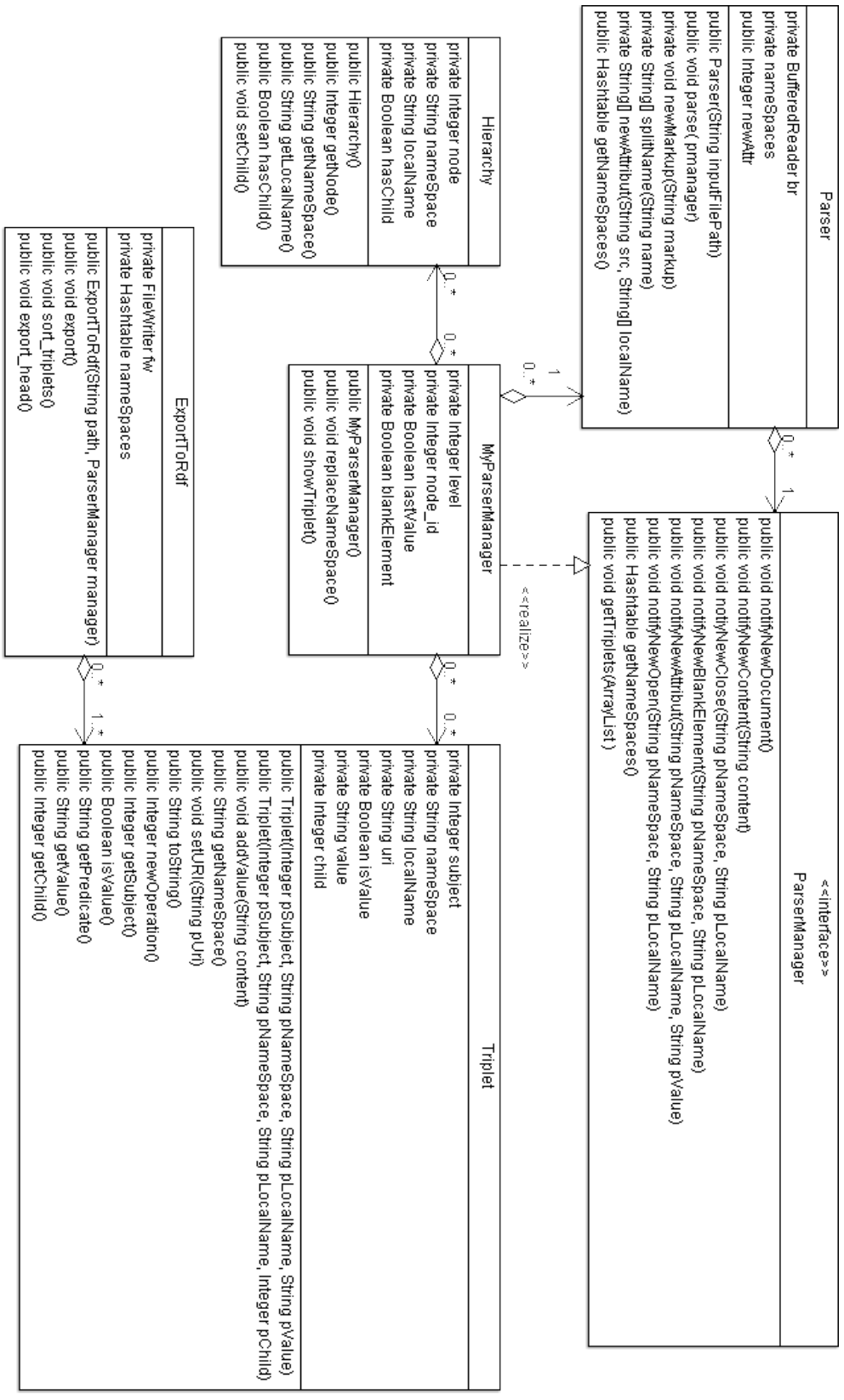

FIGURE 3.1 – Diagramme de classes UML du Parser

Cependant si il n'y a pas de perte d'informations, il n'y a pas non plus d'ajout d'informations supplémentaires telle que de l'inférence ou la gestion de la multi-occurence. Il serait également interessant d'envisager le remplacement du parser texte par un parser tel que SAX. Celui ci étant

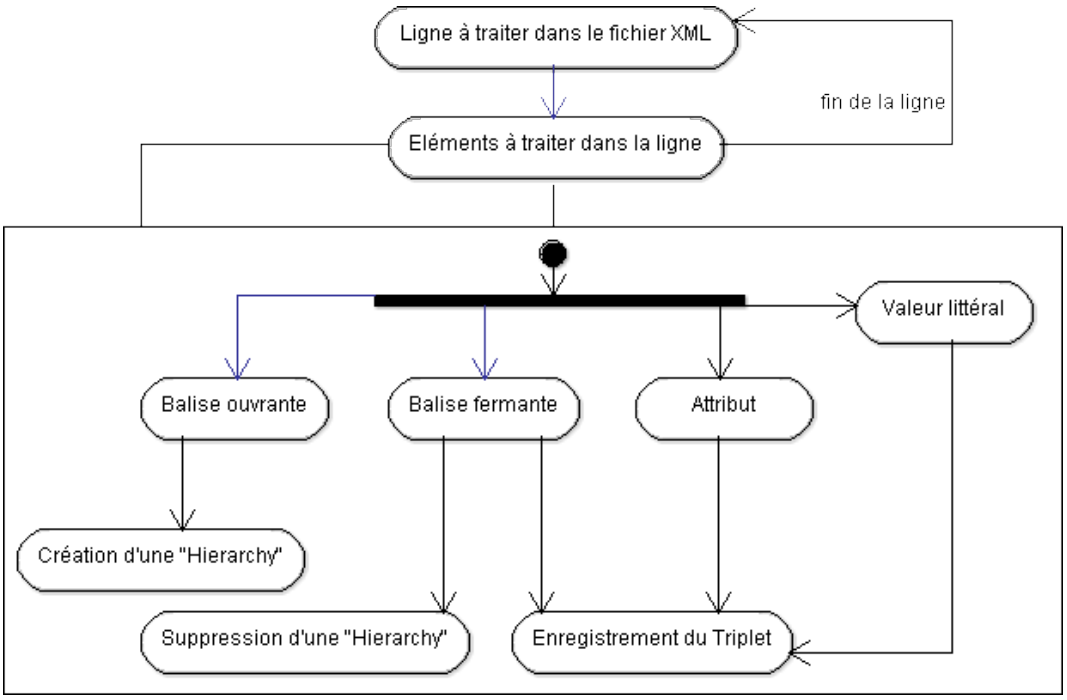

FIGURE 3.2 – Diagramme d'état du Parser

plus rapide que Dom [And10] et fonctionne sur la base d'un système d'événements comme le parser réalisé. Un parser de plus haut niveau tel que SAX ou DOM permettrait de transformer les schémas, et non d'effectuer une simple transcription. Une autre amélioration serait une meilleure gestion des namespaces. En effet dans sa version actuelle, le parser ne fait que récupérer les namespaces du fichier XML or ces namespaces font référence à un vocabulaire et à des schémas XML. En toute rigueur il faudrait utiliser un vocabulaire et des schémas spécifiques au RDF.

Ce parser est donc suffisant pour tester les performances des différentes bases de données retenues, mais demanderait quelques modifications pour une utilisation plus étendue.

3.1.2 Générateur de fiches

L'opération consiste à générer des fiches factices, elles doivent cependant être similaire du point de vue de la forme et du volume de données afin de ne pas fausser les résultats. Pour ce faire, les fiches générées sont crées à partir du modèle de méta-données sur lequel se basent les fiches tirées de MDweb. Chaque fiche contient donc les mêmes éléments de méta-données, seules les valeurs diffèrent. Le titre ainsi que les mots clés sont générés à partir d'un fichier dictionnaire qui contient une sélection de mot tirés du domaine de la géographie. Le résumé est généré à partir d'une suite de mots aléatoires et le rectangle englobant (emprise géospatiale délimitée par 4 coordonnées géographiques) à partir d'un fichier dictionnaire contenant la liste des tous les pays du monde avec leur rectangle englobant associée.

Pour créer une fiche, le générateur va donc produire les éléments soumis aux modifications aléatoires (titre, résumé, mots clés, bounding box, identifiant) soit en récupérant des éléments aléatoirement dans un dictionnaire soit en générant directement une chaîne de caractères aléatoire. Le générateur créer ensuite le fichier RDF de la nouvelle fiche sur le canevas pré-configuré. Ainsi les fiches obtenues sont factices mais valides selon les standards du W3C et ne sont pas retournées par les requêtes tests. L'une des requêtes s'effectuant par exemple sur le mot clé "service", l'ensemble des fiches factices ont étés créées de manière à ne pas contenir ce mot. Ainsi lors de la requête ces fiches factices ne feront pas parti des résultats de la requête. Lors de la montée en charge, quelque soit le nombre de fiche, le résultat des requêtes sera toujours identique.

L'utilisation de ces fiches factices va permettre de mesurer en plus du temps de réponse, la pertinence et la justesse des mesures effectuées. En effet les résultats seront prévisibles car identiques quelque soit le niveau de charge de l'outil.

3.2 Interface de test

3.2.1 Présentation de JMeter

JMeter est un outil développé en Java par la fondation Apache. Distribué sous licence libre (licence Apache 3) il est multi-plateforme. JMeter permet de réaliser de nombreux tests sur un grand nombre d'applications d'un service web à une base de données en passant par un système de messagerie. L'outil permet au travers de ses éléments d'effectuer aussi bien des tests de performance, de robustesse ou encore des tests de charge.

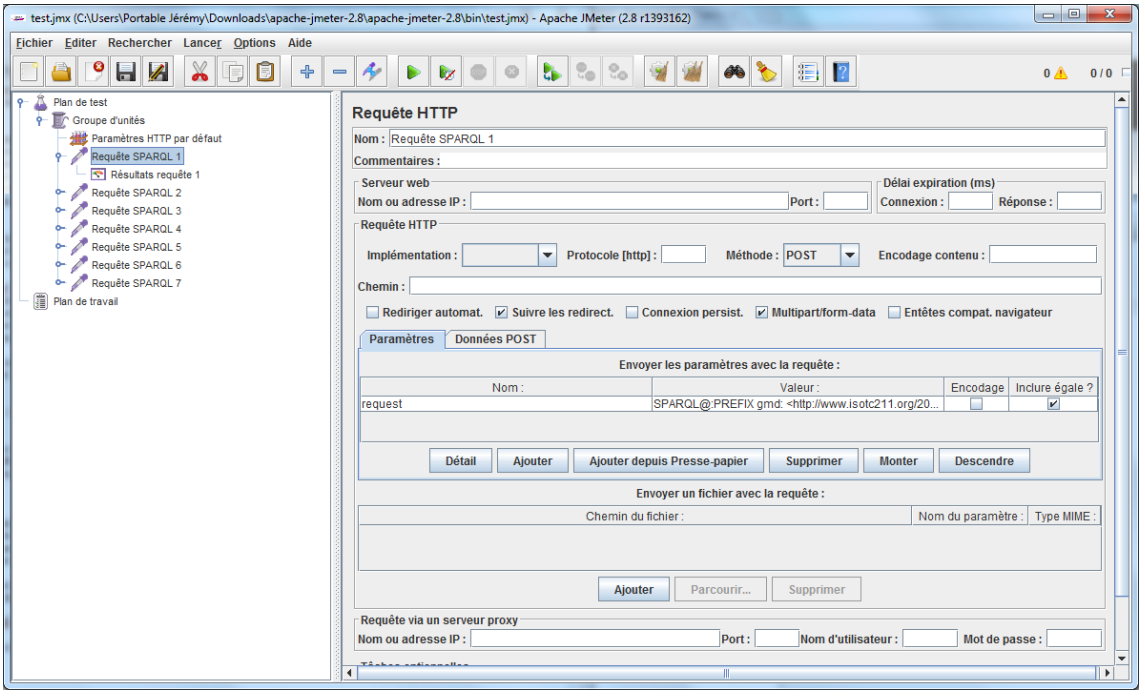

FIGURE 3.3 – Interface de JMeter configuré pour le test

La prise en main de l'outil est assez simple. Deux espaces sont définis, appelés "plans". Le plan de test contient la succession d'éléments qui serviront au test, le plan de travail contient des éléments renseignés mais non utilisés. La première étape consiste à définir **un groupe d'unité**, c'est à dire la population d'utilisateur simulée. Il est possible de définir un nombre fixe, une montée progressive en charge. Une fois ce groupe défini, le travail consiste à définir l'ensemble des actions effectuées par ces utilisateurs au travers d'actions élémentaires (accès à une page Web, requête spécifique ...). Pour faciliter la tâche des utilisateurs, des éléments de configurations sont également disponibles, ils permettent de définir les éléments nécessaires aux différentes actions (URL, identifiants par exemple). Les résultats sont alors récupérés dans des récepteurs qui peuvent être tout aussi bien un simple arbre contenant les résultats successifs ou un graphique.

JMeter offre une totale liberté pour la création des scénarios de tests. Il permet également de générer des actions conditionnelles ou aléatoires. Afin de rendre la tâche moins fastidieuse il offre la possibilité d'enregistrer un scénario de test au travers d'un proxy. Cette fonctionnalité permet de s'affranchir de l'écriture du test à la main.

3.2.2 Configuration de JMeter

Pour le test des bases de données, la création automatique via le proxy fournie par JMeter ne peut pas être utilisée car il n'y a aucune interface Web permettant d'envoyer des requêtes en SPARQL pour les bases. En effet, comme définie dans la méthodologie, le test consiste en une succession de sept requêtes exécutées plusieurs fois. Pour ce test, les sept requêtes seront exécutées dix fois, une moyenne sera ensuite réalisée avec les temps de réponse. L'opération sera réitérée pour chaque pallié de charge (pour mémoire : 320,600,1200,2500,5000,10 000, 15 000, 25 000, 50 000, 100 000).

La première étape consiste donc à créer d'un groupe d'unités dans le plan de test. Pour respecter les conditions de test, les propriétés seront réglées comme suit :

Nombre d'unités : 1

Durée de montée en charge : 1

Nombre d'itérations : 20

Cette configuration va permettre de répéter l'ensemble des sept requêtes dix fois de suite. Une fois le groupe défini il faut de choisir les paramètres HTTP par défaut.

Pour les sept requêtes, il est nécessaire de construire une requête HTTP. Pour ces requêtes, il n'est pas utile de redéfinir les informations concernant le serveur web, ceux ci ayant été définis par l'élément de configuration. Le seul élément à renseigner est la méthode, ici la méthode POST sera utilisée. Ainsi que les paramètres, ici un seul : le paramètre de nom request et contenant la requête SPARQL à exécuter. Il faut ensuite définir un récepteur. De manière simple un récepteur tableau de résultat sera utilisé, il permettra de détecter facilement les erreurs et de récupérer les temps de réponses pour chaque requête. De plus il est aisément importable dans un tableur pour être exploité ensuite.

La figure 3.3 montre le résultat final du plan de test obtenu dans JMeter, le fichier de configuration contenant le plan de test pouvant être importé dans JMeter est également disponible dans l'archive numérique.

3.2.3 Serveur Java HTTP Post

JMeter ayant été configuré pour envoyer les requêtes à un serveur web par la méthode POST, il est nécessaire d'implémenter ce serveur dans chacune des plateformes de test. Ce serveur web sera réduit au strict minimum, c'est à dire recevoir la requête POST sur un port donné, extraire la requête SPARQL transmise, la transmettre à la base de données et renvoyer le résultat. L'algorithme suivant présente la méthode principale de la classe HTTPPostServer.

Algorithme 2: Algoritme de la méthode principale du serveur HTTP Post

```
initialization;
si une demande de connexion est reçue alors
    récupération de l'entête;
    récupération de la requête;
    si requête POST alors
        tant que il y a des lignes dans la requête faire
            si paramètre request alors
                on envoie la requête à la base de données;
            fin
        fin
    fin
fin
```

La classe HTTPPostServer (dont le diagramme de classes est fourni par la figure 3.4 est une classe qui doit gérer la connexion avec l'utilisateur, c'est à dire recevoir la requête POST, la transmettre à la base de donnée et y répondre. Cette classe possède 4 attributs, le Socket qui contient le lien avec le client, les flux d'entrée et de sortie de données transmises entre le client et le serveur et l'instance gérant la base de données. Outre le constructeur qui prend en paramètre le Socket et l'instance de la classe MyDBServer (classe gérant la connexion avec la base de donnée), la classe contient deux méthodes, la méthode run() (méthode principale appelée au lancement du thread, dans laquelle la requête POST sera récupérée et transmise à la base de données et la méthode send_response() qui formate la réponse à renvoyer au client (génération des entêtes, formatage).

Ce simple serveur POST remplie ces fonctions et simule de manière satisfaisante ce que pourrait être l'utilisation dans une application web telle que MDWeb. Le fonctionnement détaillé de la classe HTTPPostServer est disponible dans la Javadoc de chaque plateforme de test disponible dans l'archive numérique.

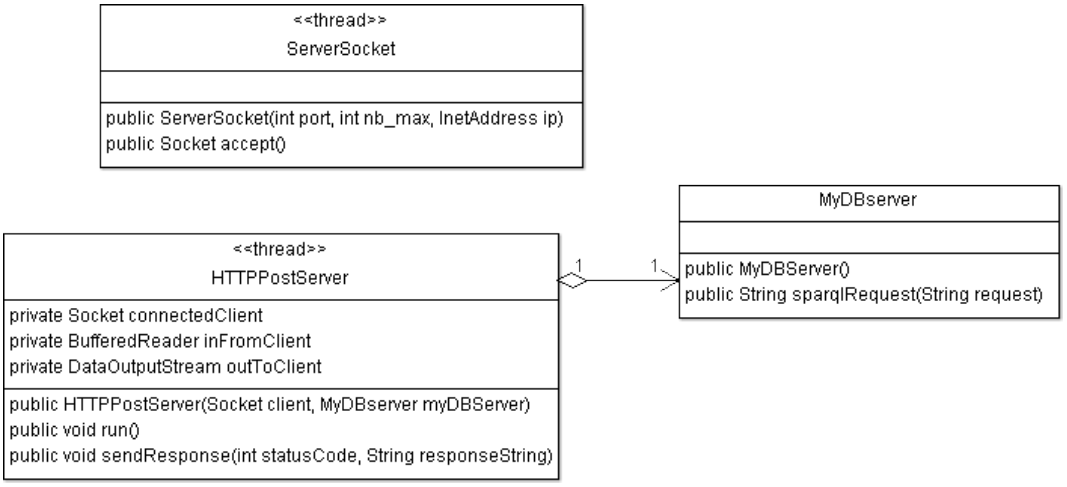

FIGURE 3.4 – Diagramme de classes UML partiel pour la classe ServerHTTPPost

3.3 Mise en place des bases de données

Pour l'ensemble des plateformes de tests (mise à part MDWeb dans sa version actuelle) la même structure a été respectée. Elles sont toutes développée en Java. Une classe Starter contenant la méthode main(). Une classe MyDBServer gérant la connection à la base de données. Une classe HTTPPostServer gérant le serveur web. Cette dernière ayant été détaillée précédemment, elle ne sera pas évoquée par la suite. La figure 3.5 montre les interactions entre ces différentes classes et la figure 3.6 correspond au diagramme de classes de la plateforme.

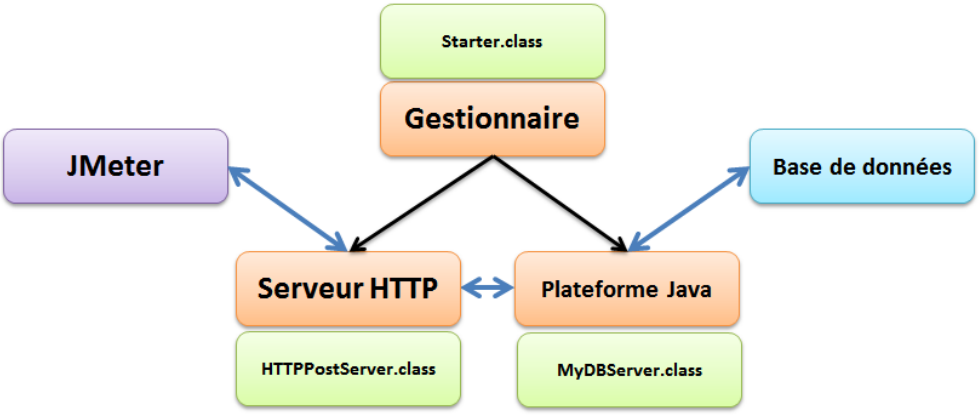

FIGURE 3.5 – Interactions entre les différents composants de la plateforme de test

Chaque plateforme de test est disponible dans l'archive numérique (code source et java doc). Pour certaines plateforme des librairies non fournies par l'éditeur sont nécessaires, elles sont également disponibles (dans la version la plus à jour disponible en novembre 2012). Dans les exemples de code présentées dans cette section, la gestion des exceptions a été omise pour plus de lisibilité.

3.3.1 OrientDB

La mise en place d'OrientDB en tant que base de données RDF est plutôt complexe car elle nécessite plusieurs librairies non fournies par l'éditeur. Pour gérer le RDF OrientDB s'appuie en effet sur les librairies OpenRDF et Tinkerpop Blueprints (qui permet de faire le lien entre OpenRDF et OrientDB). Les librairies nécessaires sont listées en annexe.

Une fois les librairies externes configurées, la plateforme est relativement simple à mettre en œuvre. La classe Starter contient une unique méthode main :

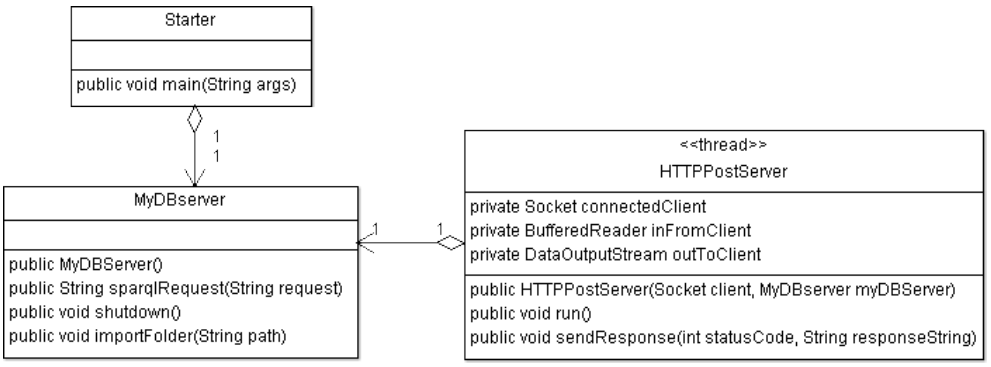

FIGURE 3.6 – Diagramme UML de classes de la plateforme de test

```
1  public static void main(String[] args) {
2      MyDBServer myDBServer = new MyDBServer("jeu1");
3      myDBServer.importFolder("rdf");
4      ServerSocket Server;
5      try {
6          Server = new ServerSocket (5000, 10, InetAddress.getByName("127.0.0.1"));
7          while(true) {
8              Socket connected = Server.accept();
9                  (new HTTPPOSTServer(connected, myDBServer)).start();
10         }
11     } catch (IOException e) {
12         e.printStackTrace();
13     }
14 }
```

La ligne 2 correspond à l'instantiation de la classe gérant la base de données (MyDBServer). Ce constructeur prend en paramètre le nom de la base de données. Ce constructeur va démarrer le serveur OrientDB (le serveur est embarqué) et créer le graph RDF avec l'intermédiaire de la classe GraphSail de Tinkerpop Blueprints. Une fois ces deux opérations effectuées le serveur est fonctionnel.

La ligne 3 correspond à l'import des fiches RDF (contenue dans le dossier "rdf") dans la base de données. Pour cela la méthode importFolder() de la classe MyDBServer va créer les correspondances entre OrientDB et OpenRDF (à l'aide de classes et méthodes fournies par OpenRDF). Ensuite chaque fichier est importé successivement dans la base de données, la librairie OpenRDF permettant le parsing et l'ajout de fichiers RDF. La suite de la fonction main() correspond à la gestion du serveur HTTP Post détaillé plus haut.

Les requêtes SPARQL reçues par le serveur HTTP Post sont traitées par la méthode sparqlRequest() de la classe MyDBServer. Cette méthode prend en paramètre la requête SPARQL et retourne une chaîne de caractères contenant le résultat de la requête. Le code suivant présente la manière dont sont traitées les requêtes SPARQL :

```
1  String result = "";
2  SPARQLParser parser = new SPARQLParser();
3  CloseableIteration<? extends BindingSet, QueryEvaluationException> sparqlResults;
4  ParsedQuery query = parser.parseQuery(queryString, "http://mesfiches.com");
5  sparqlResults = sc.evaluate(query.getTupleExpr(), query.getDataset(), new ↩
       EmptyBindingSet(), false);
6  while (sparqlResults.hasNext()) {
7      BindingSet temp = sparqlResults.next();
8      result = result + temp;
9  }
```

La ligne 2 du bloc de code précédent correspond à l'instantiation de l'interpréteur de requête SPARQL fournie par la libraire OpenRDF. A la ligne 4 la requête est interprétée afin de pouvoir être exécutée par OpenRDF. A la ligne suivante, la requête est exécutée. Le résultat est retournée sous la forme d'une collection avec un itérateur. Cette collection est parcourue via l'itérateur et le résultat est stocké dans une chaîne de caractère qui sera renvoyée en résultat.

Pour lancer la plateforme de test, il suffit donc d'exécuter le programme Java, en ayant au préalable placé les fiches à insérer dans le dossier "rdf" ou, si la base est déjà constituée, d'avoir mis en commentaire la ligne 3. Les requêtes SPARQL peuvent ensuite être envoyée via la méthode

GET à l'adresse : http ://ip_du_serveur :5000.

3.3.2 MongoDB

La mise en place de la plateforme MongoDB est relativement similaire à celle d'OrientDB, car encore une fois ce sont les librairies Tinkerpop Blueprints et OpenRDF qui vont être utilisées. Les librairies à importer sont donc sensiblement les mêmes et sont également listées en annexe.

Pour fonctionner, la plateforme nécessite également les packages com.tinkerpop.blueprints.impl.mongodb et packages com.tinkerpop.blueprints.impl.mongodb qui ne sont pas disponibles en .jar mais uniquement en source. Il suffit donc de copier ces deux packages dans le dossier src du projet.

La structure de la classe Starter est la même, la méthode main est identique à l'exception de la ligne 2 où le constructeur de MyDBserver() ne prend plus de paramètre. Le code suivant correspond au constructeur de la classe MyDBServer, ce code est identique à celui utilisé pour la plateforme OrientDB, mise à part la ligne 4 qui correspond à l'instantiation de la base de données, et est donc spécifique à celle ci.

```
1  public MyDBServer() {
2      graph = null;
3      try {
4          graph = new MongoDBGraph("127.0.0.1",27017);
5          sail = new GraphSail<MongoDBGraph>(graph);
6          sail.initialize();
7          sc = sail.getConnection();
8          vf = sail.getValueFactory();
9          System.out.println("[MongoDB] MongoDB server is now running.");
10     } catch (SailException e) {
11         e.printStackTrace();
12     }
13 }
```

La ligne 5 correspond à la création du graph RDF via GraphSail (librairie Tinkerpop Blueprints). C'est cette classe (utilisée également avec OrientDB) qui permet réellement de gérer le RDF, elle réalise la liaison entre la structure de la base (graph, document) et RDF (structure en triplets). Les lignes suivantes constituent l'initialisation des classes permettant d'effectuer des opérations sur la base (ajout, suppression, requête).

Contrairement à OrientDB, MongoDB ne propose pas de serveur embarqué, il est donc nécessaire de lancer le serveur au préalable. Ce serveur est disponible sur toutes les plateformes (Windows, Mac, Linux) et est téléchargeable directement depuis le site de l'éditeur. Il suffit ensuite de lancer le programme Java et il devient possible d'envoyer des requêtes SPARQL au serveur web.

3.3.3 Jena

La mise en place d'une plateforme de test pour Jena est totalement différente de celle des deux précédents outils. En effet pour OrientDB et MongoDB il s'agissait de trouver un moyen de gérer le RDF par dessus une base de données. Alors que Jena est un outil gérant le RDF auquel il faut ajouter une composante permettant de rendre les données persistantes. Deux grandes solutions sont possibles, passer par une base de donnée relationnelle telle que PostgreSQL ou utiliser le module JenaTDB. La première solution est relativement simple à mettre en place mais nécessite tout de même d'installer un serveur PostgreSQL et utiliser les drivers JDBC pour accéder à la base de données depuis Java. Pour cette plateforme c'est la seconde solution qui sera utilisée, étant une fonctionnalité de Jena, elle ne nécessite aucune installation autre.

Les librairies nécessaires sont réduites par rapports aux solutions précédentes, et sont toutes fournies sur le site de Appache Jena :

La classe Starter et la fonction main() sont strictement identiques à celles d'OrientDB, de même que le serveur HTTP, seul la classe MyDBServer gérant le serveur diffère. Le constructeur MyDBServer(), qui prend en paramètre le nom de la base de donnée est plus simple :

```
1  public MyDBServer(String db_name) {
2       String directory = "data/" + db_name ;
3       dataset = TDBFactory.createDataset(directory) ;
4       model = dataset.getNamedModel("http://mesfiches.com/"+db_name);
5  }
```

La ligne 3 correspond au chargement des données dans un container : Dataset. En effet Jena ne gère pas l'accès à une base de donnée à proprement parlé, elle charge la totalité du graphe RDF dans la mémoire vive et effectue ensuite les requêtes dessus. Cette méthode offre un avantage certain de rapidité, mais également plusieurs inconvénients, dont la nécessité d'allouer énormément de mémoire vive à la machine virtuelle. La ligne 4 correspond à la construction du graphe à partir du container. Ce graphe pourra ensuite être parcouru facilement.

L'import des fiches est également différent. Il se réalise en deux étapes, tout d'abord les fiches sont parsés par Jena et ajoutées au graphe en mémoire vive, puis il est nécessaire d'enregistrer ce graphe via JenaTDB. Le code suivant provient de la classe importFolder() de la classe MyDBServer, qui gère cet import.

```
1  Model tempmodel = ModelFactory.createDefaultModel();
2  File folder = new File(path);
3  String [] filesList = folder.list();
4  for(int i = 0; i < filesList.length;i++)    {
5      String extension = filesList[i].substring(filesList[i].lastIndexOf(".") + 1);
6      String name = filesList[i].substring(0,filesList[i].lastIndexOf("."));
7      if(extension.compareTo("rdf")==0)
8          tempmodel.read(new FileReader("rdf/"+name+".rdf"),"http://mesfiches.com");
9  }
10 dataset.begin(ReadWrite.WRITE) ;
11 dataset.addNamedModel("http://mesfiches.com/"+db_name, tempmodel);
12 dataset.commit() ;
13 dataset.end() ;
```

Les trois premières lignes correspondent à la récupération de la liste des fichiers contenus dans le répertoire passé en paramètre. La première boucle "for" importe chaque fiche (contenu dans un fichier .rdf). La condition à la ligne 7 permet de vérifier qu'il s'agit bien d'un fichier .rdf (cette condition est particulièrement utile lorsque la plateforme est déployée sous MacOS, où plusieurs fichiers supplémentaires sont créés par l'OS). La ligne 8 quand à elle constitue le parsing et l'import de la fiche dans un graphe RDF temporaire (instancié à la ligne 1). Le reste du code correspond à l'enregistrement des données via JenaTDB. La ligne 10 correspond au début de la transaction sur le Dataset. La ligne 11 correspond à la requête d'insertion et la ligne 12 au commit(). Jena gère donc un système relativement semblable au système transactionnel des bases de données relationnelles (si une erreur se produit lors de la requête, avant le commit(), rien n'est enregistré).

La gestion des requêtes SPARQL (via le serveur HTTP Post puis traitées par la méthode sparqlRequest() de MyDBServer) est très simple via Jena. La ligne 1 du code suivant correspond au parsing (interprétation) de la requête SPARQL, les lignes 2 et 3 à son exécution. Il suffit ensuite de retourner le résultat sous forme de chaîne de caractère (la méthode toString() est implémentée nativement dans JenaTDB).

```
1  public String sparqlRequest(String request) {
2      Query query = QueryFactory.create(request);
3      QueryExecution qe = QueryExecutionFactory.create(query, model);
4      ResultSet results = qe.execSelect();
5      qe.close();
6      return results.toString();
7  }
```

En utilisant JenaTDB, l'intégralité de la plateforme est autonome, il suffit donc de lancer le programme Java pour pouvoir requêter via l'interface web. Cependant, il faut veiller à allouer suffisament de mémoire à la machine virtuelle (environ 1Go de RAM pour pouvoir traiter 15000 fiches) via le paramètre -Xmx1024m (pour allouer 1024Mo à la JVM).

3.3.4 MDWeb

La mise en place de MDWeb est relativement simple et expliquée en détail sur le site internet qui en fait la distribution (www.mdweb-project.org) [MDW12].

Tout d'abord MDWeb est une Web Application donc le fichier .war est disponible et doit être téléchargé sur le site de l'éditeur. Ensuite cette application nécessite un serveur d'application pour être exécuté, pour cette plateforme de test, c'est le logiciel Apache Tomcat qui a été retenu. Il est également nécessaire de récupérer le JDK (Java Developpement Kit). Deux extensions doivent être ajoutées au JDK : JAI et JAI-Image-IO. Il est ensuite nécessaire d'installer une base de données. Pour ce test PostgreSQL sera utilisé.

Une fois Tomcat lancé, il suffit de déployer la Web Archive via le panneau d'administration. L'installation de MDWeb étant ensuite rapide. Le requètage se fera ensuite via le CSW de MDWeb. Cependant MDWeb ne propose pas de requête SPARQL. Cette plateforme servira uniquement de référence aux autres tests.

3.3.5 OrientDB - Nouvelle structure de stockage

Aux vues des lenteurs constatées durant les premiers tests avec la base de données OrientDB peuplées des fiches RDF telles que générées, une seconde plateforme, prenant en compte le modèle de la norme a été mise en place. Sur la première plateforme chaque fiche a été enregistrée telle quelle, c'est à dire que chaqu'une contenait le modèle de méta-donnée (cf Figure 2.17) alors que sur la seconde plateforme le modèle de norme est enregistré une fois, et lors de l'enregistrement des fiches uniquement les instances et valeurs sont enregistrées (cf Figure 2.18).

La figure ci dessous illustre la simplification qu'apporte la description du modèle pour l'élément Keyword pour l'élément Keyword d'une fiche de méta-données :

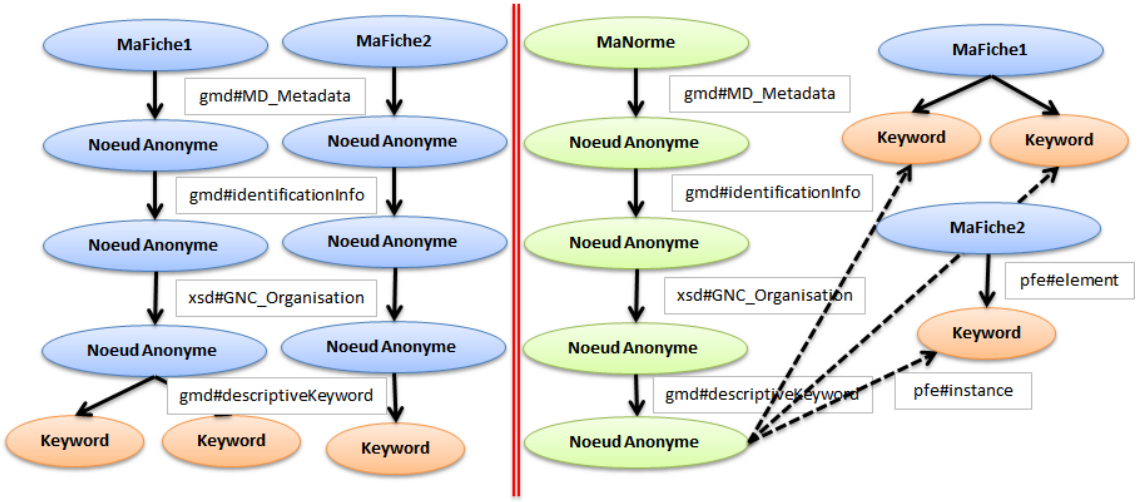

FIGURE 3.7 – Comparaison des deux plateformes OrientDB

La structure de la plateforme est donc la même que pour OrientDB, mis à par la méthode importFolder() de la classe MyDBServer. Tout d'abord les fichiers importés ne sont plus au même format, sur les plateformes précédentes (OrientDB, MongoDB et Jena, les fichiers importés étés des fichiers XML/RDF, pour cette plateforme l'import se fait à partir de fichier N3. Les fichiers N3 contiennent uniquement les triplets (pas de format XML) et sont donc plus simple à importer. Pour ce faire, il a été nécessaire d'ajouter ce format de sortie au ParserXML et au générateur de fiches.

Lors de la création de la base, on ajoute le schéma RDF de la norme. Pour les tests, ce schéma sera limité aux éléments utilisés par les fiches utilisées. Cet ajout se fait à l'aide d'une classe GeonetCabSchema. Cette classe va ajouter le schéma à la base de donnée, mais également conserver le graphe RDF. Ainsi lors de l'ajout d'un triplet, il suffira de passer ce triplet en paramètre d'une méthode needInstance() de la classe GeonetCabSchema, pour déterminer si ce triplet est un triplet correspondant au schéma de la norme (par exemple : NoeudAnonyme1 : gmd#MD_METADATA : NoeudAnonyme2) ou si il est nécessaire d'enregistrer ce noeud comme étant un élément de la fiche (par exemple : NoeudAnonyme3 : gmd#abstract : NoeudAnonyme4). Cette opération est présentée

dans l'algorithme suivant :

Algorithme 3: Algorithme ajout triplet

```
Données : Triplet
initialization;
si le sujet est un noeud présent dans la base alors
    on enregistre le triplet dans la base;
sinon
    si le triplet doit être enregistré alors
        on enregistre le triplet reliant la fiche au noeud attribut;
        on enregistre le triplet reliant le noeud de la norme au noeud attribut;
    fin
fin
```

Lors de l'ajout d'une fiche, l'arborescence du graphe RDF est donc parcourue. Chaque branche est testée avec la méthode needInstance(). Dès que la méthode de test renvoie vraie, alors un premier noeud est créé. La création de ce noeud entraine l'ajout de deux triplets dans la base de données. Tout d'abord un premier entre le noeud issu du schéma et un noeud anonyme (avec pour prédicat : http ://mesfiches.com#instance), et un second triplet entre le noeud représentant la fiche et le noeud anonyme (avec pour prédicat : http ://mesfiches.com#element). Par la suite tous les noeuds fils de l'arborescence sont enregistrés.

Le fonctionnement de la plateforme est ensuite similaire à celle expliquée plus haut. Cependant il est nécessaire de réécrire les requêtes SPARQL qui ont légèrement changées avec cette modification de la structure. Avec cette plateforme, le nombre de triplets stockés (et donc requêtés lors des tests) est environ divisé par 2, pour un nombre équivalent de fiches stockés.

Chapitre 4

Mesures et résultats

Sommaire

4.1 Conditions de tests

4.1.1 Environnement de tests

Les tests ont été effectuées avec deux ordinateurs portables de marque Asus. Les deux machines (clients et serveurs ayant été reliées par un cable Ethernet RJ45). Sur les deux machines les processus ont été limités à leur strict minimum (pas de programme tiers). Les configurations des deux machines sont les suivantes :

Serveur :

Système d'exploitation Windows Edition Familiale Prémium (64bits) SP1

Version de Java JDK 1.6

Mémoire allouée à la JVM 1024Mo

Processeur Intel Core i7 2,3GHz

Mémoire vive 8Go DDR3

Carte réseau Carte ethernet 100MB/s

Client :

Système d'exploitation Windows Edition Intégrale (32bits) SP1

Version de Java JRE 1.7

Mémoire allouée à la JVM 1024Mo

Processeur Intel Core i3 2,2GHz

Mémoire vive 4Go DDR3

Carte réseau Carte ethernet 100MB/s

4.1.2 Description de la mesure

Pour réaliser ce test de performances, chaque technologie a été soumise à sept requêtes itérées vingts fois. Ces sept requêtes, de complexité graduelle, représentent un panorama des requêtes possibles et courantes sous MDWeb.
Liste des requêtes :

- Requête lexicale avec le mot clé "service"
- Requête lexicale avec le mot clé "service" ET le type de ressource "Software"
- Requête lexicale avec les mots clés "service" ET "agriculture"
- Requête géospatiale (rectangle englobant)
- Requête lexicale avec le mot clé "service" ET un rectangle englobant
- Requête lexicale avec le mot clé "service" ET le type de ressource "Software" ET un rectangle englobant
- Requête lexicale avec les mots clés "service" ET "agriculture" ET un rectangle englobant

Lors des tests, ce sont les temps de réponses (donnés en millisecondes par JMeter) qui constitueront le coeur de la mesure. La figure 4.1 montre les résultats retournés par JMeter. Les valeurs extrêmes sont ensuite supprimées et une moyenne de ces temps de réponses est effectuées. Ce sont ces moyennes des temps de réponses qui sont utilisés dans les graphiques suivant.

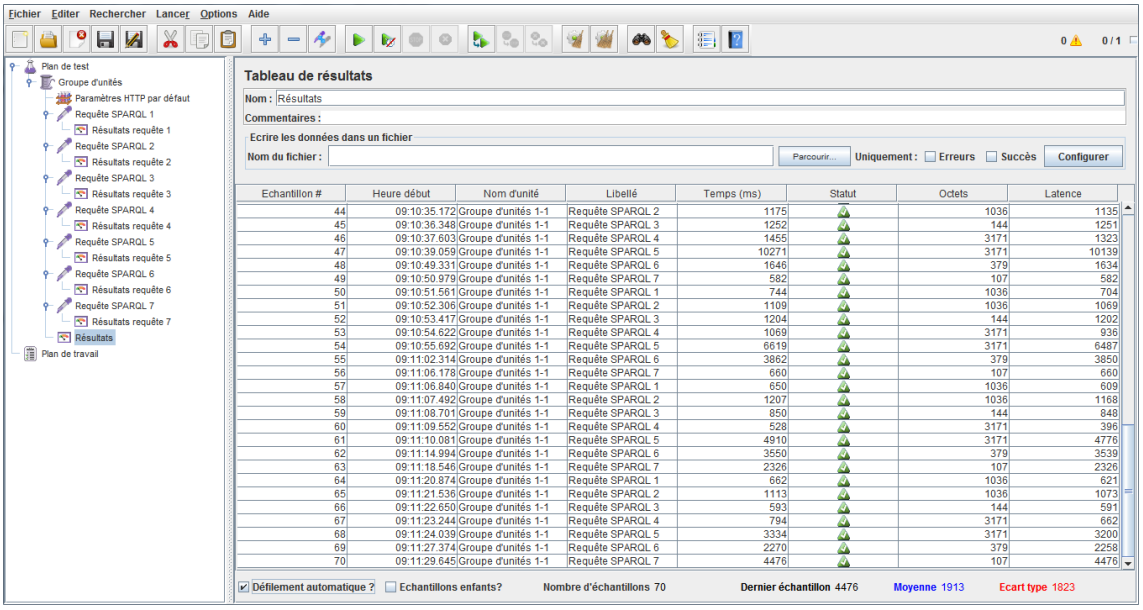

FIGURE 4.1 – Résultats obtenus sous JMeter

4.2 Résultats

4.2.1 Mesures

Deux mesures ont été réalisées pour ces tests de performance. Tout d'abord un test de justesse, résultats correctes par rapports aux résultats totaux, et de précision, résultats correctes par rapports aux résultats attendus, dont les résultats sont présentés dans le tableau 4.1. Ces mesures consistent en une comparaison des résultats retournées par les requêtes envoyées aux bases de données comparés aux résultats retournées par les requêtes envoyées à MDWeb. Les résultats obtenus ne sont pas optimaux, en effet certains résultats qui devraient être retournés ne le sont pas. En revanche tous les résultats retournées sont correct.

Cependant l'ensemble des technologies testées retournes les mêmes résultats, ces erreurs proviennent donc probablement d'un problème d'optimisation et de conception des requêtes SPARQL (fournies en annexe). La comparaison ne pouvant se faire sur ce point, le temps de réponse, deuxième mesure effectuée, sera détaillée par la suite.

TABLE 4.1 – Mesure de justesse et de rappel des différentes technologies

Technologie	Justesse	Rappel
OrientDB	100%	92%
MongoDB	100%	92%
Jena	100%	92%

MongoDB

MongoDB se révèle être extrêmement lent pour les requêtes SPARQL avec des requêtes pouvant nécessiter plus de 200 secondes pour s'exécuter sur un jeu de 320 fiches. A la vue de ces premiers résultats, la montée en charge n'a pas été réalisée, la gestion de 320 fiches étant déjà trop long. Le tableau 4.2 donne cependant les résultats obtenus pour un jeu de 320 fiches.

TABLE 4.2 – Temps de réponse moyen (ms) de MongoDB

Nb. fiches	Requête 1	Requête 2	Requête 3	Requête 4	Requête 5	Requête 6	Requête 7
320	16239	59315	15849	8049	206327	16906	15541

Jena

Jena (ici c'est JenaTBD qui est testé, c'est à dire la version TripletStore de la technologie) charge en mémoire l'ensemble des index ainsi qu'une partie des données. On obtient donc d'excellents résultats présentés par le tableau 4.3. Cependant il est nécessaire d'attribuer une quantité très important de mémoire vive à la machine virtuelle Java. En effet avec 1Go de mémoire il n'est pas possible de gérer plus de 15 000 fiches. De plus la mise en cache des données demande un certain temps. Ce temps de monté en cache est donnée par le tableau 4.4

OrientDB

Les résultats obtenus avec OrientDB sont mitigés, ils semblent meilleurs que ceux obtenus sous MongoDB mais demeurent tout de même trop importants. Le grahique présenté par la Figure 4.2 présente les résultats obtenus pour la montée en charge. En abscisse sont représentés le nombre de fiches, correspondant à la montée en charge. En ordonnée est représenté le temps de réponse moyen pour chaque requête.

On observe un dépassement rapide du seuil acceptable pour le temps de réponse. De plus la requête 5 (lexicale et géospatiale) dépasse clairement les autres requêtes. Encore une fois cet erreur provient probablement d'une mauvaise optimisation de la requête SPARQL.

Une autre plateforme pour OrientDB, utilisant une autre architecture a également été testée. Les résultats des tests réalisées sur cette plateforme sont présentés dans le graphique en Figure 4.3. On

TABLE 4.3 – Temps de réponse moyen (ms) de Jena pour différents niveaux de charge

Nb. fiches	Requête 1	Requête 2	Requête 3	Requête 4	Requête 5	Requête 6	Requête 7
320	11	13	12	13	13	14	13
600	11	12	11	13	12	13	13
1200	11	11	11	12	12	13	13
2500	11	11	12	13	12	13	13
5000	11	11	12	12	12	13	13

TABLE 4.4 – Temps de montée en cache (ms) de Jena pour différents niveaux de charge

Nb. fiches	320	600	1200	2500	5000
Temps	742	721	723	736	734

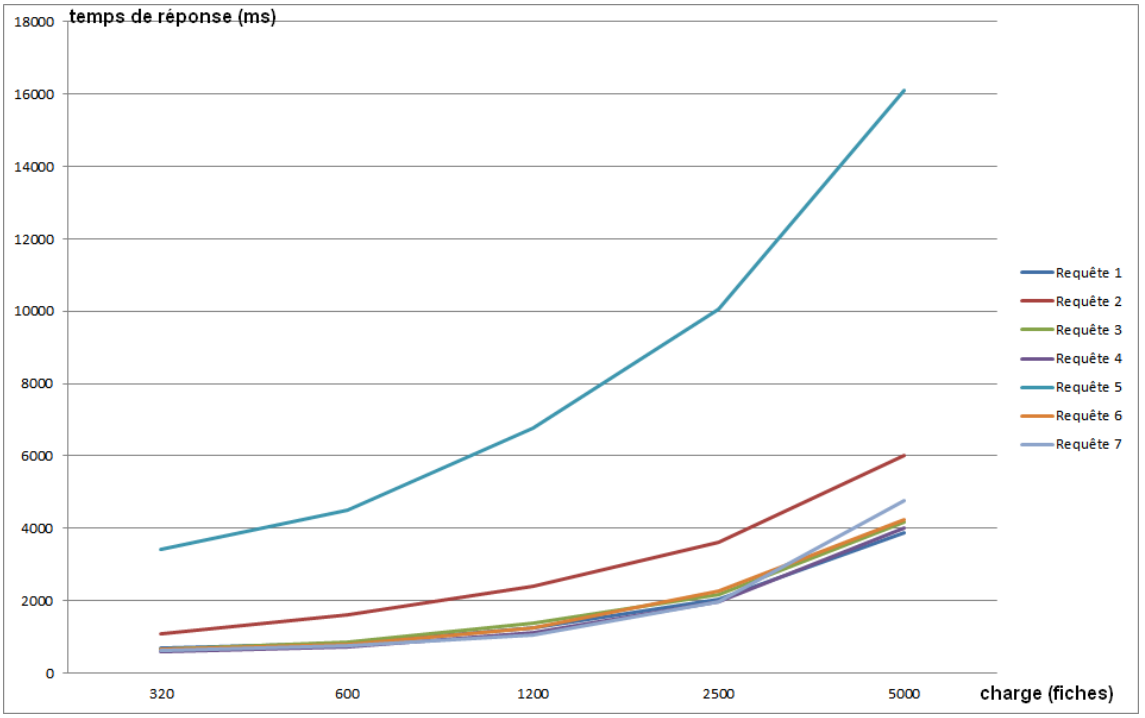

FIGURE 4.2 – Graphique du temps de réponse en fonction du nombre de fiche sous OrientDB

observe une diminution du temps de réponse à niveau de charge égale. Cependant, l'augmentation du temps de réponse est identique. De plus on observe toujours un temps trop long pour la requête 5, ce qui confirme la piste de la mauvaise optimisation de la requête.

4.2.2 Mesures comparées

Le graphique présenté par la figure 4.4 montre les différents temps de réponses pour chaque outil et pour chaque requête à un niveau de charge donnée, à savoir 320 fiches. Il est important de noter que l'échelle des ordonnées est une échelle logarithmique. Un grande différence de temps de réponse est très nettement observable entre les différentes technologies. Et toutes souffrent d'un cruel manque de performances vis à vis de la technologie existante.

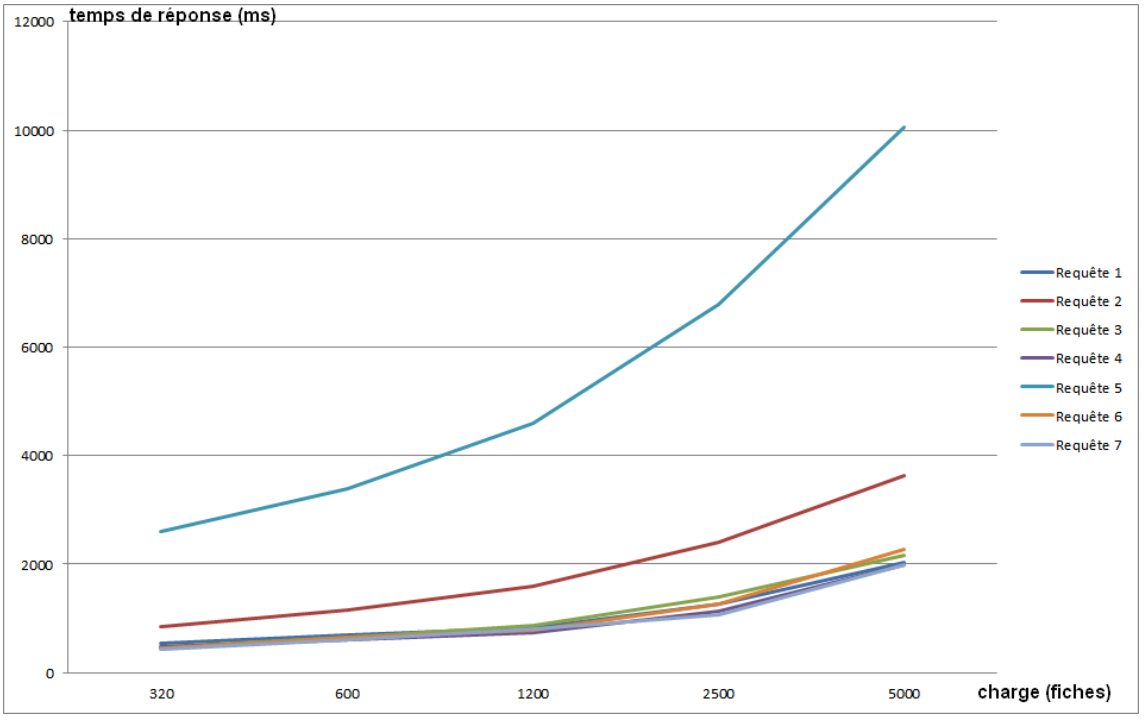

FIGURE 4.3 – Graphique du temps de réponse en fonction du nombre de fiche sous OrientDB avec nouvelle structure

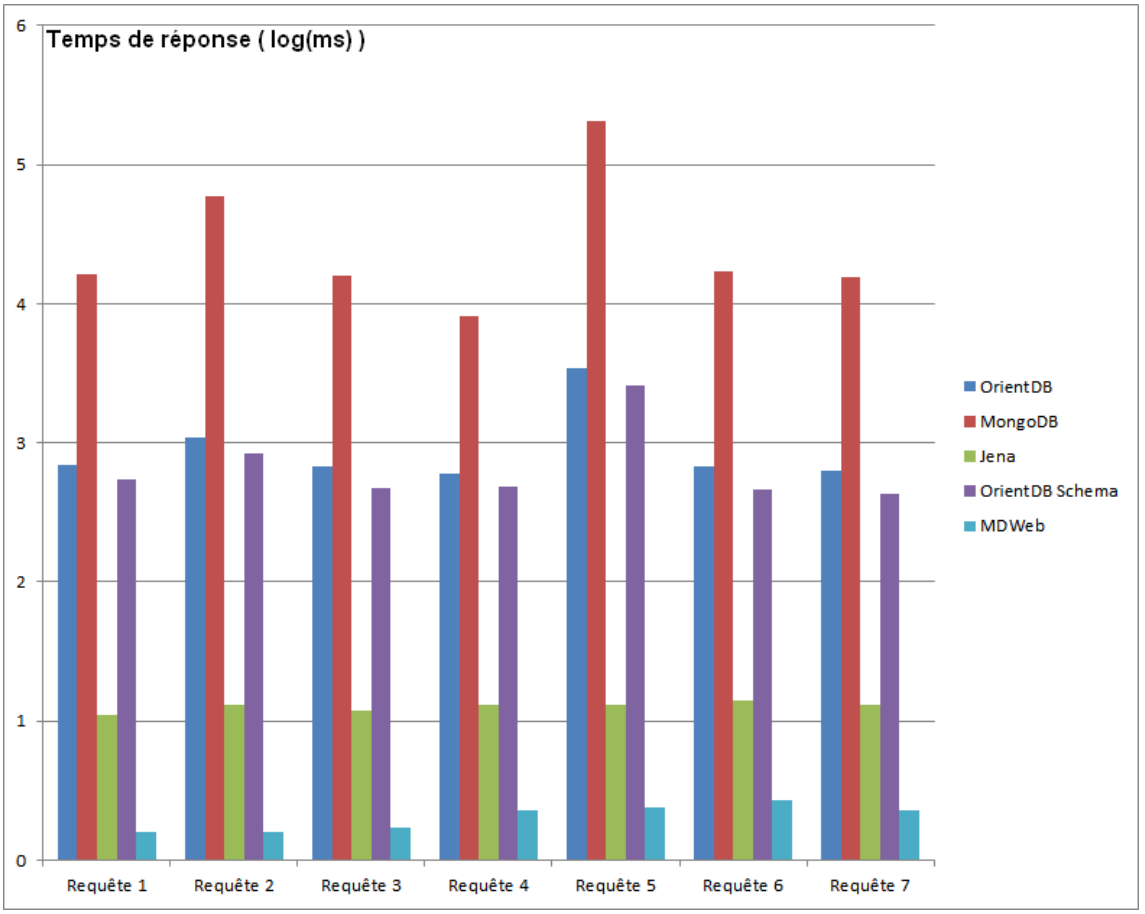

FIGURE 4.4 – Graphique comparatif du temps de réponse pour 320 fiches pour chaque base

Conclusion

L'objectif de ce stage consistait en une étude des solutions NoSQL comme technologie de stockage des méta-données en RDF pour l'outil libre de catalogage et de localisation MDWeb. Cet objectif est en partie atteint. Un état de l'art détaillé des différentes technologies disponibles a été réalisé, un banc d'essai a été conçu et mise en œuvre sur trois solutions de stockage RDF : OrientDB, une base de données orienté graphe, MongoDB, une base orienté document, Jena dans une configuration TripleStore et l'outil MDweb comme outil de référence des mesures. Ce banc d'essai avait pour but de mesurer les performances des différentes solutions choisies. Cependant aucune technologie viable, en l'état, n'a pu être dégagée.

En effet, les résultats obtenus sont mitigés, bien que la précision des mesures soit tout à fait acceptable, les erreurs semblant provenir d'un manque d'optimisation des requêtes. Les mesures des temps de réponse montre elles de réelle lacunes, avec des temps de réponse dépassant souvent la seconde. Cependant l'outil le plus prometteur parmi ceux testés, reste OrientDB.

Il est également important de noter que ces technologies ont été testés avec une implémentation de RDF, via une libraire externe. Il est envisageable de tester à nouveau OrientDB avec une autre plateforme. De plus, l'implémentation RDF utilisée ne faisait à aucun moment appel aux modèles. Ainsi la conversion des fiches XML ISO 19139 réalisée pourrait grandement être améliorée par la prise en compte des modèles. Les plateformes mise en place souffrent également d'une mauvaise gestion des index dû aux librairies externes qui ne semblent pas utiliser les index des bases de données efficacement.

Durant les tests il a été envisagé de tester OrientDB en implémentant un modèle. Cette optimisation s'est révélée prometteuse. Outre le gain de performance au niveau des requêtes, ce passage est nécessaire pour pouvoir gérer différentes normes et l'alignement entre ces normes, or ce point est l'un des motifs du passage en RDF potentiel de MDWeb. De plus pour pouvoir utiliser un raisonneur et effectuer de l'inférence de manière efficace il est nécessaire de posséder une représentation par modèle. D'autre pistes d'optimisation sont également ouvertes, telle que l'optimisation des requêtes ou l'ajout d'index.

Finalement, la solution du tout RDF semble compromise. Même avec l'utilisation d'un Triple Store. En effet la structure des fiches de méta-données impose un nombre très élevé de triplets et une arborescence trop importante. Il serait possible d'adopter technologie telle que Jena avec une architecture in memory (dans la mémoire vive), tel que SAP HANA. Solution trop coûteuse qui limiterait le panel d'utilisateurs de MDWeb.

Perspectives

Si il n'a pas été possible de dégager clairement une technologie à implémenter, plusieurs pistes de réflexion et de recherche ont étés mises au jour.

Une alternative au passage de la solution de stockage relationnelle vers un stockage sous forme de triplets serait l'utilisation d'un outil assurant la correspondance des entités. En effet ce type d'approche permettrait de conserver la base actuelle (base de données relationnelle) en y ajoutant ce composant apportant la possibilité d'effectuer des requêtes SPARQL. Le mapping permettrait donc de faire cohabiter deux technologies offrant ainsi les avantages des deux technologies, à savoir la performance et la robustesse des bases relationnelles mais également la richesse du langage RDF.

Cependant ce genre d'outil de mapping est en général gourmand en ressources. Et même si le W3C semble aujourd'hui favoriser le développement de ce genre de technologies, ce qui promet un développement de ces technologies dans un avenir proche, les performances resteraient moindre que celle d'une base de données relationnelle.

Durant ce stage, il a été envisagé d'utiliser D2RQ qui est un des outils les plus utilisés pour

effectuer le mapping de bases relationnelles. Cependant nous nous sommes retrouvés confrontés au problème de l'élaboration du fichier de mapping, qui s'avère très complexe pour la structure de la base de données de MDWeb.

Il est également envisageable de n'effectuer ce mapping que partiellement. En effet une opération de partition de graphe pourrait être effectuée en amont avec des critères de choix stricts comme la délimitation d'une zone géographique ou une thématique. L'opération consisterait alors à mapper les résultats obtenus sous la forme d'un graphe, qui pourrait ensuite être traité in memory via Jena par exemple.

Ce procédé pourrait permettre de profiter des richesses du RDF sur un jeu réduit de données éliminant ainsi les écueils de performances ou de coût liés à la quantité importante de mémoire nécessaire. Cependant pour être pertinente elle nécessiterait une étude conséquente sur la partition de graphe. En effet cette partition doit être précise. Il n'est pas envisageable de passer à coté de certains résultats, la partition doit donc contenir l'ensemble des résultats susceptibles d'intéresser l'utilisateur, cependant pour des raisons de performance il ne doit contenir que le minimum d'information possible. La difficulté d'une telle approche est donc de trouver un équilibre entre ces deux contraintes.

Une dernière alternative consisterait à l'utilisation d'une base graphe telle que OrientDB, non pas pour stocker du RDF mais pour stocker les fiches avec les modèles. Cette solutions permettrait de stocker efficacement ces modèles, de pouvoir facilement gérer l'alignement de normes. Cependant pour pouvoir être requêter en SPARQL, il est nécessaire d'implémenter un interpréteur SPARQL. Le stockage n'est alors plus en RDF, ce qui reviendrait à effectuer un mapping, mais sur une base graphe plutôt qu'une base relationnelle.

Annexes

Annexe 1 - Jeu de requêtes SPARQL

Requête 1 -

PREFIX gmd : <http ://www.isotc211.org/2005/gmd#>
PREFIX gco : <http ://www.isotc211.org/2005/gco#>
PREFIX ns11 : <http ://www.mdweb-project.org/files/xsd#>
SELECT ?fiche WHERE { { ?fiche gmd :MD_Metadata ?noeud11 . ?noeud11 gmd :identificationInfo ?noeud12 . ?noeud12 ns11 :GNC_Organisation ?noeud13 } .
{ { ?noeud13 gmd :citation ?noeud14 . ?noeud14 gmd :CI_Citation ?noeud15 . ?noeud15 gmd :title ?noeud16 . ?noeud16 gco :CharacterString ?title FILTER regex(?title, "service", "i") }
UNION { ?noeud13 gmd :abstract ?noeud24 . ?noeud24 gco :CharacterString ?abstract FILTER regex(?abstract, "service", "i") }
UNION { ?noeud13 gmd :descriptiveKeywords ?noeud34 . ?noeud34 gmd :MD_Keywords ?noeud35 . ?noeud35 gmd :keyword ?noeud36 . ?noeud36 gco :CharacterString ?keyword FILTER regex(?keyword, "service", "i") } } }
GROUP BY ?fiche

Requête 2 -

PREFIX gmd : <http ://www.isotc211.org/2005/gmd#>
PREFIX gco : <http ://www.isotc211.org/2005/gco#>
PREFIX ns11 : <http ://www.mdweb-project.org/files/xsd#>
SELECT ?fiche WHERE { { ?fiche gmd :MD_Metadata ?noeud11 . ?noeud11 gmd :identificationInfo ?noeud12 . ?noeud12 ns11 :GNC_Organisation ?noeud13 } .
{ { ?noeud13 gmd :citation ?noeud14 . ?noeud14 gmd :CI_Citation ?noeud15 . ?noeud15 gmd :title ?noeud16 . ?noeud16 gco :CharacterString ?title FILTER regex(?title, "service", "i") }
UNION { ?noeud13 gmd :abstract ?noeud24 . ?noeud24 gco :CharacterString ?abstract FILTER regex(?abstract, "service", "i") }
UNION { ?noeud13 gmd :descriptiveKeywords ?noeud34 . ?noeud34 gmd :MD_Keywords ?noeud35 . ?noeud35 gmd :keyword ?noeud36 . ?noeud36 gco :CharacterString ?keyword FILTER regex(?keyword, "service", "i") } } .
{ ?noeud13 ns11 :resourceType ?neud44 . ?noeud44 ns11 :codeListValue ?type FILTER regex(?type, "Software", "i") } }
GROUP BY ?fiche

Requête 3 -

PREFIX gmd : <http ://www.isotc211.org/2005/gmd#>
PREFIX gco : <http ://www.isotc211.org/2005/gco#>
PREFIX ns11 : <http ://www.mdweb-project.org/files/xsd#>
SELECT ?fiche WHERE { { ?fiche gmd :MD_Metadata ?noeud11 . ?noeud11 gmd :identificationInfo ?noeud12 . ?noeud12 ns11 :GNC_Organisation ?noeud13 } .
{ { ?noeud13 gmd :citation ?noeud14 . ?noeud14 gmd :CI_Citation ?noeud15 . ?noeud15 gmd :title ?noeud16 . ?noeud16 gco :CharacterString ?title FILTER (regex(?title, "service", "i") && regex(?title, "Agriculture", "i")) }
UNION { ?noeud13 gmd :abstract ?noeud24 . ?noeud24 gco :CharacterString ?abstract FILTER (

regex(?abstract, "service", "i") && regex(?abstract, "Agriculture", "i")) }
UNION { ?noeud13 gmd :descriptiveKeywords ?noeud34 . ?noeud34 gmd :MD_Keywords ?noeud35
. ?noeud35 gmd :keyword ?noeud36 . ?noeud36 gco :CharacterString ?keyword FILTER (regex(?keyword,
"service", "i") && regex(?keyword, "Agriculture", "i")) } } }
GROUP BY ?fiche

Requête 4 -

PREFIX xsd : <http ://www.w3.org/2001/XMLSchema#>
PREFIX gmd : <http ://www.isotc211.org/2005/gmd#>
PREFIX gco : <http ://www.isotc211.org/2005/gco#>
PREFIX ns11 : <http ://www.mdweb-project.org/files/xsd#>
SELECT ?fiche WHERE { { ?fiche gmd :MD_Metadata ?noeud11 . ?noeud11 gmd :identificationInfo ?noeud12 . ?noeud12 ns11 :GNC_Organisation ?noeud13 . ?noeud13 gmd :extent ?noeud14 . ?noeud14 gmd :EX_Extent ?noeud15 . ?noeud15 gmd :geographicElement ?noeud16 . ?noeud16 gmd :EX_GeographicBoundingBox ?bbox } .
{ { ?bbox gmd :northBoundLatitude ?north1 . ?north1 gco :Decimal ?north2 FILTER(xsd :decimal(?north2) < 34.401 && xsd :decimal(?north2) > -28.204) }
UNION { ?bbox gmd :southBoundLatitude ?south1 . ?south1 gco :Decimal ?south2 FILTER(xsd :decimal(?south2) > -28.204 && xsd :decimal(?south2) < 34.401) }
UNION { ?bbox gmd :westBoundLongitude ?west1 . ?west1 gco :Decimal ?west2 FILTER(xsd :decimal(?west2) > -108.984 && xsd :decimal(?west2) < -26.016) }
UNION { ?bbox gmd :eastBoundLongitude ?east1 . ?east1 gco :Decimal ?east2 FILTER(xsd :decimal(?east2) < -26.016 && xsd :decimal(?east2) > -108.984) } } }
GROUP BY ?fiche

Requête 5 -

PREFIX xsd : <http ://www.w3.org/2001/XMLSchema#>
PREFIX gmd : <http ://www.isotc211.org/2005/gmd#>
PREFIX gco : <http ://www.isotc211.org/2005/gco#>
PREFIX ns11 : <http ://www.mdweb-project.org/files/xsd#>
SELECT ?fiche WHERE { { ?fiche gmd :MD_Metadata ?noeud11 . ?noeud11 gmd :identificationInfo ?noeud12 . ?noeud12 ns11 :GNC_Organisation ?noeud13 } .
{ ?noeud13 gmd :extent ?noeud14 . ?noeud14 gmd :EX_Extent ?noeud15 . ?noeud15 gmd :geographicElement ?noeud16 . ?noeud16 gmd :EX_GeographicBoundingBox ?bbox } .
{ { ?bbox gmd :northBoundLatitude ?north1 . ?north1 gco :Decimal ?north2 FILTER(xsd :decimal(?north2) < 34.401 && xsd :decimal(?north2) > -28.204) }
UNION { ?bbox gmd :southBoundLatitude ?south1 . ?south1 gco :Decimal ?south2 FILTER(xsd :decimal(?south2) > -28.204 && xsd :decimal(?south2) < 34.401) }
UNION { ?bbox gmd :westBoundLongitude ?west1 . ?west1 gco :Decimal ?west2 FILTER(xsd :decimal(?west2) > -108.984 && xsd :decimal(?west2) < -26.016) }
UNION { ?bbox gmd :eastBoundLongitude ?east1 . ?east1 gco :Decimal ?east2 FILTER(xsd :decimal(?east2) < -26.016 && xsd :decimal(?east2) > -108.984) } } .
{ ?noeud13 ns11 :resourceType ?neud44 . ?noeud44 ns11 :codeListValue ?type FILTER regex(?type, "SuccessStory", "i") } }
GROUP BY ?fiche

Requête 6 -

PREFIX xsd : <http ://www.w3.org/2001/XMLSchema#>
PREFIX gmd : <http ://www.isotc211.org/2005/gmd#>
PREFIX gco : <http ://www.isotc211.org/2005/gco#>
PREFIX ns11 : <http ://www.mdweb-project.org/files/xsd#>
SELECT ?fiche WHERE { { ?fiche gmd :MD_Metadata ?noeud11 . ?noeud11 gmd :identificationInfo ?noeud12 . ?noeud12 ns11 :GNC_Organisation ?noeud13 } .
{ { ?noeud13 gmd :citation ?noeud14 . ?noeud14 gmd :CI_Citation ?noeud15 . ?noeud15 gmd :title ?noeud16 . ?noeud16 gco :CharacterString ?title FILTER regex(?title, "service", "i") }

```
UNION { ?noeud13 gmd :abstract ?noeud24 . ?noeud24 gco :CharacterString ?abstract FILTER re-
gex( ?abstract, "service", "i") }
UNION { ?noeud13 gmd :descriptiveKeywords ?noeud34 . ?noeud34 gmd :MD_Keywords ?noeud35
. ?noeud35 gmd :keyword ?noeud36 . ?noeud36 gco :CharacterString ?keyword FILTER regex( ?keyword,
"service", "i") } } .
{ ?noeud13 gmd :extent ?noeud44 . ?noeud44 gmd :EX_Extent ?noeud45 . ?noeud45 gmd :geogra-
phicElement ?noeud46 . ?noeud46 gmd :EX_GeographicBoundingBox ?bbox } .
{ { ?bbox gmd :northBoundLatitude ?north1 . ?north1 gco :Decimal ?north2 FILTER(xsd :deci-
mal( ?north2) < 34.401 && xsd :decimal( ?north2) > -28.204) } UNION { ?bbox gmd :south-
BoundLatitude ?south1 . ?south1 gco :Decimal ?south2 FILTER(xsd :decimal( ?south2) > -28.204
&& xsd :decimal( ?south2) < 34.401) }
UNION { ?bbox gmd :westBoundLongitude ?west1 . ?west1 gco :Decimal ?west2 FILTER(xsd :de-
cimal( ?west2) > -108.984 && xsd :decimal( ?west2) < -26.016) }
UNION { ?bbox gmd :eastBoundLongitude ?east1 . ?east1 gco :Decimal ?east2 FILTER(xsd :deci-
mal( ?east2) < -26.016 && xsd :decimal( ?east2) > -108.984) } } }
GROUP BY ?fiche
```

Requête 7 -

```
PREFIX xsd : <http ://www.w3.org/2001/XMLSchema#>
PREFIX gmd : <http ://www.isotc211.org/2005/gmd#>
PREFIX gco : <http ://www.isotc211.org/2005/gco#>
PREFIX ns11 : <http ://www.mdweb-project.org/files/xsd#>
SELECT ?fiche WHERE { { ?fiche gmd :MD_Metadata ?noeud11 . ?noeud11 gmd :identificatio-
nInfo ?noeud12 . ?noeud12 ns11 :GNC_Organisation ?noeud13 } .
{ { ?noeud13 gmd :citation ?noeud14 . ?noeud14 gmd :CI_Citation ?noeud15 . ?noeud15 gmd :title ?noeud16
. ?noeud16 gco :CharacterString ?title FILTER ( regex( ?title, "service", "i") && regex( ?title,
"Agriculture", "i") ) }
UNION { ?noeud13 gmd :abstract ?noeud24 . ?noeud24 gco :CharacterString ?abstract FILTER (
regex( ?abstract, "service", "i") && regex( ?abstract, "Agriculture", "i") ) }
UNION { ?noeud13 gmd :descriptiveKeywords ?noeud34 . ?noeud34 gmd :MD_Keywords ?noeud35
. ?noeud35 gmd :keyword ?noeud36 . ?noeud36 gco :CharacterString ?keyword FILTER ( regex( ?keyword,
"service", "i") && regex( ?keyword, "Agriculture", "i") ) } }.
{ ?noeud13 gmd :extent ?noeud44 . ?noeud44 gmd :EX_Extent ?noeud45 . ?noeud45 gmd :geogra-
phicElement ?noeud46 . ?noeud46 gmd :EX_GeographicBoundingBox ?bbox } .
{ { ?bbox gmd :northBoundLatitude ?north1 . ?north1 gco :Decimal ?north2 FILTER(xsd :deci-
mal( ?north2) < 34.401 && xsd :decimal( ?north2) > -28.204) }
UNION { ?bbox gmd :southBoundLatitude ?south1 . ?south1 gco :Decimal ?south2 FILTER(xsd :de-
cimal( ?south2) > -28.204 && xsd :decimal( ?south2) < 34.401) }
UNION { ?bbox gmd :westBoundLongitude ?west1 . ?west1 gco :Decimal ?west2 FILTER(xsd :de-
cimal( ?west2) > -108.984 && xsd :decimal( ?west2) < -26.016) }
UNION { ?bbox gmd :eastBoundLongitude ?east1 . ?east1 gco :Decimal ?east2 FILTER(xsd :deci-
mal( ?east2) < -26.016 && xsd :decimal( ?east2) > -108.984) } } }
GROUP BY ?fiche
```

Annexe 2 - Tableau comparatif des systèmes NoSQL

Types	Clé Valeur	Clé Valeur	Colonnes	Colonnes
Nom	Riak	Voldemort	Hbase	Cassandra
Editeur	Basho	LinkedIn	Apache	Facebook
Licence	Apache License 2	Apache License 2	Apache License 2	Apache License 2
Utilisateurs	Mozilla, Github	LinkedIn	Facebook (2010)	Facebook, Twitter
Query Langage	HTTP Rest	API dédiée, HTTP Rest	CQLsh, API dédiée	HTTP Rest, API dédiée
Documentation	Faible	Faible	Correcte	Importante
Java	Non	Oui	Oui	Oui
Communauté	Restreinte	Restreinte	Moyenne	Moyenne
Orientation CAP	AP	AP	CP	AP
Support	Non	Non	Oui	Oui
Complexité	Faible	Faible	Importante	Importante
Version	1.2	0.9	0.94.1	1.1.4
Types	Documents	Documents	Graphes	Graphes
Nom	CouchDB	MongoDB	Neo4j	OrientDB
Editeur	Apache	10Gen	NeoTechnology	OrientTechnologies
Licence	License Apache 2	GNU Affero GPL	GNU GPL v3	Apache License 2
Utilisateurs	Apple, BBC	NY Times, SourceForge	Adobe , Cisco	
Query Langage	API dédiée	HTTP Rest, API dédiée	API dédiée, Gremlin, SPARQL (v 1.0)	API dédiée, Gremlin, SQL, SPARQL (v 1.1)
Documentation	Importante	Importante	Bonne	Correcte
Java	Oui	Oui	Oui	Oui
Communauté	Importante	Importante	Importante	Moyenne
Orientation CAP	AP	CP	CP	CP
Support	Non	Oui	Oui	Non
Complexité	Importante	Moyenne	Moyenne	Moyenne
Version	1.2	2.2.0	1.8	1.1.0

Types	Graphes	Graphes	Triple Store	Triple Store	Triple Store
Nom	HyperGraphDB	FlockDB	Jena	AlegroGraph	Sesame
Editeur	Kobriw Software	Twitter	Apache	Franz Inc	OpenDB
Licence	LGPL	Apache License 2	Apache Licence 2	Closed source	BSD-3-Clause
Utilisateurs		Twitter	HP		
Query Langage	API dédiée, Gremlin, SQL	Ruby client	SPARQL (V 1.0 et V 1.1) , SeRQL	SPARQLl(V 1.0)	SPARQL(V 1.0)
Documentation	Bonne	Correcte	Bonne	Bonne	Bonne
Java	Oui	Non	Oui	oui	Oui
Communauté	Moyenne	Moyenne	Importante	Moyenne	Importante
Orientation CAP	AP	AP	AP	AC	AP
Support	Non	Non	Oui	oui	Oui
Complexité	Importante	Faible	Moyenne	?	Moyenne
Version	1.1	1.8.5	2.6.4	4.6	2.6.9

Annexe 3 - Librairies nécessaires à la plateforme de test

OrientDB

- blueprints-core-2.1.0-SNAPSHOT.jar
- blueprints-graph-sail-2.1.0.jar
- blueprints-orient-graph-2.1.0-SNAPSHOT.jar
- orientdb-core-1.1.0.jar
- orient-commons-1.1.0.jar
- pipes-2.0.0-SNAPSHOT.jar
- sesame-model-2.6.9.jar
- sesame-util-2.6.9.jar
- sesame-sail-api-2.6.9.jar
- sesame-sail-inferencer-2.6.9.jar
- sesame-query-2.6.9.jar
- sesame-queryalgebra-evaluation-2.6.9.jar
- sesame-queryalgebra-model-2.6.9.jar
- sesame-queryparser-api-2.6.9.jar
- sesame-queryparser-sparql-2.6.9.jar
- sesame-rio-api-2.6.9.jar
- sesame-rio-rdfxml-2.6.9.jar
- sesame-repository-api-2.6.9.jar
- sesame-repository-sail-2.6.9.jar
- slf4j-api-1.7.0.jar
- slf4j-simple-1.7.0.jar
- common-1.6.jar

MongoDB

- blueprints-core-2.1.0-SNAPSHOT.jar
- blueprints-graph-sail-2.1.0.jar
- mongo-2.9.1.jar
- sesame-model-2.6.9.jar
- sesame-util-2.6.9.jar
- sesame-sail-api-2.6.9.jar
- sesame-sail-inferencer-2.6.9.jar
- sesame-query-2.6.9.jar
- sesame-queryalgebra-evaluation-2.6.9.jar
- sesame-queryalgebra-model-2.6.9.jar
- sesame-queryparser-api-2.6.9.jar
- sesame-queryparser-sparql-2.6.9.jar
- sesame-rio-api-2.6.9.jar
- sesame-rio-rdfxml-2.6.9.jar
- sesame-repository-api-2.6.9.jar
- sesame-repository-sail-2.6.9.jar
- slf4j-api-1.7.0.jar
- slf4j-simple-1.7.0.jar
- common-1.6.jar

Jena

- jena-core-2.7.3.jar
- jena-tdb-0.9.3.jar
- jena-arq-2.9.3.jar
- jena-iri-0.9.3.jar
- log4j-over-slf4j-1.7.0.jar
- slf4j-api-1.7.0.jar
- xercesImpl-2.10.0.jar

– xml-apis-1.4.01.jar

Bibliographie

[ALS09] C. Anderson, J. Lehnardt, and N. Slater. *Couchdb : The definitive guide*. O'Reilly Media, 2009.

[And10] S. Andrès. Autour de java & xml, 2010.

[Apa12a] Apache. Apache cassandra 1.1 manual. http ://www.datastax.com/docs/1.1/index, Consulté le 28 août 2012.

[Apa12b] Apache. Apache hbase reference guide. http ://hbase.apache.org/book.html, Consulté le 28 août 2012.

[Bas12] Basho. Riak manual. http ://wiki.basho.com, Consulté le 28 août 2012.

[Bre00] E.A. Brewer. Towards robust distributed systems. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, volume 19, pages 7–10, 2000.

[Bur11] G. Burd. Nosql. *Login*, 36, 2011.

[CD10] K. Chodorow and M. Dirolf. *MongoDB : the definitive guide*. O'Reilly Media, Inc., 2010.

[CDG+08] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable : A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2) :4, 2008.

[Cos12] A. Costes. Une base nosql : Cassandra. http ://www-igm.univ-mlv.fr/ dr/X-POSE2010/Cassandra, Consulté le 28 août 2012.

[CST+10] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[DHJ+07] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo : amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[dSedS04] Observatoire du Sahara et du Sahel. *Organisation, fonctionnement et méthodes de ROSELT/OSS*. ROSELT/OSS, 2004.

[Fiq09] M. Fiquière. Devoxx – jour 1 – nosql avec hbase. http ://blog.xebia.fr/2009/11/18/devoxx-jour-1-nosql-avec-hbase/, 2009.

[Fiq10a] M. Fiquière. Nosql europe : Bases de données clé-valeur et riak. http ://blog.xebia.fr/2010/04/26/nosql-europe-bases-de-donnees-cle-valeur-et-riak/, 2010.

[Fiq10b] M. Fiquière. Nosql europe : Bases de données graphe et neo4j. http ://blog.xebia.fr/2010/05/03/nosql-europe-bases-de-donnees-graphe-et-neo4j/, 2010.

[Fiq10c] M. Fiquière. Nosql europe : Bases de données orientées documents et mongodb. http ://blog.xebia.fr/2010/04/30/nosql-europe-bases-de-donnees-orientees-documents-et-mongodb/, 2010.

[Fou11] A. Foucret. Nosql, une nouvelle approche du stockage et de la manipulation des données. *Livre Blanc Smile*, 2011.

[Fra12] T. Francart. Rdf : Sesame, jena, comparaison des fonctionnalités. http ://francart.fr/rdf-sesame-jena-comparaison-des-fonctionnalites/, Consulté le 4 septembre 2012.

[Gar12] L. Garulli. Orientdb - select the right model. http ://fr.slideshare.net/lvca/orientdb-document-or-graph-select-the-right-model, Consulté le 29 août 2012.

[IE10] Geomatys IRD EspaceDev. *Manuel utilisateur de MDWeb*. IRD Espace Dev, 2010.

[Ior10] B. Iordanov. Hypergraphdb : a generalized graph database. *Web-Age Information Management*, pages 25–36, 2010.

[Ipp09] B. Ippolito. Drop acid and think about data. In *Talk at Pycon on March 2009*, 2009.

[ISO03] ISO 19115 : Geographic information - Metadata, 2003.

[ISO07] ISO 19139 : Geographic information - Metadata - XML schema implementation, 2007.

[ISO11] ISO. http ://www.isotc211.org/. *site internet*, January 2011.

[JCD08] Thérèse LIBOUREL Jean-Christophe DESCONNETS. *MDWEB– catalogage et localisation de ressources environnementales*. IRD Espace Dev, 2008.

[Kre09] J. Kreps. Project voldemort. 2009.

[Lai09] E. Lai. No to sql? anti-database movement gains steam. http ://www.computerworld.com/s/article/9135086/No_to_SQL_Anti_database_movement_gains_steam_
2009.

[Lin12] LinkedIn. Project voldemort : A distributed database. http ://project-voldemort.com/voldemort, Consulté le 28 août 2012.

[LM09] A. Lakshman and P. Malik. Cassandra : A structured storage system on a p2p network. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 47–47. ACM, 2009.

[MBD04] B. McBride, D. Boothby, and C. Dollin. An introduction to rdf and the jena rdf api. *Retrieved August*, 1 :2007, 2004.

[MDW12] MDWeb. Quick start : Installation de mdweb version 2.3. http ://www.mdweb-project.org/files/documentation/2011.09/Quickstart_2.3.pdf, Consulté le 23 novembre 2012.

[Mor12] M. Morello. Orientée colonnes ? http ://michaelmorello.blogspot.fr/2012/06/orientee-colonne.html, Consulté le 28 août 2012.

[MT92] Z. Mihalic and N. Trinajstic. A graph-theoretical approach to structure : Property relationships. *Journal of chemical education*, 69(9) :701–712, 1992.

[Per12] M. Perham. Comparing document-oriented databases. http ://www.mikeperham.com/2009/09/01/comparing-document-oriented-databases/, Consulté le 28 août 2012.

[Pie08] C. Pierkot. *Gestion de la Mise à Jour de Données Géographiques Répliquées*. PhD thesis, Université Paul Sabatier-Toulouse III, 2008.

[PM06] D.A. Papa and I.L. Markov. Hypergraph partitioning and clustering. *Approximation algorithms and metaheuristics. CRC, Boca Ratan*, 2006.

[Pol09] J.T. Pollock. *Semantic Web for dummies*. For Dummies, 2009.

[SC310] SC32/WG2. http ://metadata-stds.org/. *site internet*, 2010.

[Sri12] A. Srivastava. Nosql, newsql and mdm. http ://cdi-mdm.blogspot.fr/2011/07/nosql-newsql-and-mdm.html, Consulté le 4 septembre 2012.

[SSK11] C. Strauch, U.L.S. Sites, and W. Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 2011.

[Suv] D. Suvee. Rdf data in neo4j - the tinkerpop story. http ://java.dzone.com/news/rdf-data-neo4j-tinkerpop-story.

[Té] Téluq. Introduction au rdf. http ://www.teluq.ca/inf6450/mod5/intrordf.xhtml.

[Tam12] D. Tam. Facebook processes more than 500 tb of data daily. http ://news.cnet.com/8301-1023_3-57498531-93/facebook-processes-more-than-500-tb-of-data-daily, 2012.

[Tea12] Neo4j Team. *The Neo4j Manual*. Neo Technology, 2012.

[Tiw11] S. Tiwari. *Professional NoSQL*. Wrox, 2011.

[Twi12] Twitter. Introducing flockdb. http ://engineering.twitter.com/2010/05/introducing-flockdb.html, Consulté le 27 août 2012.

[VMZ+10] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database : a data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, page 42. ACM, 2010.

[Whi10] T. White. *Hadoop : The definitive guide*. Yahoo Press, 2010.