Académie de Montpellier
Université Montpellier II
Sciences et Techniques du Languedoc

MÉMOIRE DE STAGE DE MASTER 2

effectué à la société **Intactile DESIGN**

Spécialité : **Informatique (AIGLE)**

Définition de modèles de connaissance pour la gestion et l'exploitation de données spatio-temporelles

par **Mojdeh SOLTAN MOHAMMADI**

Date de soutenance : **Juillet 2014**

Sous la direction de :
Isabelle MOUGENOT
Christophe FAGOT

Remerciements

Je souhaitais adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire.

Je tiens à remercier mes encadrants Isabelle Mougenot et Christophe Fagot pour leurs soutien, conseils, encouragements et disponibilité qui m'ont été indispensables pour achever ce travail. Qu'ils trouvent ici ma sincère reconnaissance pour tous ce qu'ils ont pu m'apporter.

Je remercie aussi Mme Thérèse Libourel pour sa disponibilité et ces précieux conseils qui m'ont été fructueux.

J'adresse également mes remerciements à l'équipe de l'entreprise IntactileDESIGN pour la bonne humeur qui a rendu agréable la période de mon stage.

Je n'oublie pas mes parents pour leur contribution, leur soutien et leur patience.

J'exprime ma gratitude aux membres du jury pour avoir bien voulu juger mon travail et pour le temps qu'ils ont accordé à la lecture de mon rapport.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amis, qui m'ont toujours soutenue et encouragée au cours de la réalisation de ce mémoire.

Merci à tous et à toutes.

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION

Contexte de l'étude

De multiples jeux de données sont aujourd'hui, rendus accessibles sur le Web. Ces jeux de données sont caractérisés en tout premier lieu par le domaine d'intérêt qui en a permis l'acquisition, à l'exemple du domaine de l'urbanisme ou de la gestion des territoires. De manière complémentaire, ils sont également caractérisés par une double dimension : spatiale et temporelle, qu'il convient de prendre en charge à leur juste valeur, dans les systèmes de gestion de données. Si nous prenons comme exemple le domaine du transport aérien ou maritime, cette double dimension va se révéler essentielle. Un vol aérien va ainsi se matérialiser par un temps de vol, un point de départ et d'arrivée ainsi qu'un espace traversé. Il va alors s'agir de proposer l'organisation et le traitement des informations relatives au temps et à l'espace, de manière à construire les services attendus par les compagnies qui privilégient la rentabilité et la sécurité des lignes aériennes, ou encore par les voyageurs qui privilégient le confort, le coût et le temps de leur voyage.

Si les volumes de données peuvent être très importants, les utilisateurs du système également peuvent être très nombreux, et exprimer des demandes d'accès et de traitement des données en simultané. De plus, les données et les utilisateurs peuvent être localisés à différents endroits de la planète. Les données peuvent être complexes et de natures diverses au regard du domaine d'application. Les besoins en terme d'usage des données peuvent couvrir aussi bien des traitements statistiques à l'attention des décideurs que des restitutions d'informations visuelles à destination d'utilisateurs finaux. Les technologies qui vont supporter la gestion et le traitement des données, doivent permettre de fournir aux utilisateurs des réponses fiables dans un temps raisonnable. Si nous nous intéressons à un contexte en particulier comme celui de la supervision de moyens de transport à l'échelle internationale, les systèmes à mettre en place sont, par la force des choses, distribués : le problème de passage à l'échelle (ou scalabilité) se pose alors. Ces systèmes devront prendre en charge, de la manière la plus naturelle possible, les aspects relatifs à la spatialité et à la temporalité portant sur de gros volume de données possiblement hétérogènes, et potentiellement exploitables par de nombreux utilisateurs.

Objectifs du travail présenté

Notre étude se concentre sur la gestion et l'exploitation de données possiblement chrono-référencées et géo-référencées dans un contexte de forte sollicitation par les utilisateurs. Les nouveaux besoins, en matière d'accès et d'utilisation de l'information, en particulier sur le Web, ont fait évoluer les manières d'envisager la gestion des données. Ainsi des systèmes de gestion de données, nommés NoSQL, pour Not Only SQL, vont faciliter la gestion de gros volume de données distribuées, destinées à satisfaire les besoins en information de larges communautés d'usagers. Des systèmes dits New SQL proposent une adaptation des systèmes relationnels aux contraintes imposées par le passage à l'échelle. Des systèmes de gestion des triplets appelés *triplestore* vont faciliter le stockage de données au format ouvert RDF, et par extension permettre d'enrichir ces données au travers de

l'application de mécanismes d'interprétation exprimés dans les langages RDFS et OWL. Nous nous intéresserons à l'ensemble de ces systèmes pour en évaluer les performances d'accès, de consultation et d'interprétation sur de larges volumes de données dans des environnements multi-utilisateurs, tout en gardant à l'esprit les facilités d'usage pour la mise en œuvre de ces solutions. Notre réflexion s'inscrit dans une approche dite "grande masse de données" (aussi appelée *Bigdata* ou *Datamasse*) avec une considération particulière pour les données spatio-temporelles. Ainsi, les différents systèmes de gestion de données seront comparés sur leur capacité à proposer des facilités pour le spatial comme pour le temporel. Dans un deuxième temps, nous porterons notre effort sur la définition d'un modèle de connaissances approprié à la prise en charge des dimensions spatiales et temporelles et ce, pour un contexte d'étude en particulier qui relève de la navigation marchande. Nous privilégierons la réutilisation de modèles de données déjà existants et éprouvés pour le spatial comme pour le temporel. L'objectif est de rester au plus près des standards pour des raisons d'interopérabilité des données.

Plan du manuscrit

Le manuscrit est organisé comme suit :
Dans un premier chapitre, nous détaillons le modèle de données OpenStreetMap et nous présentons un premier jeu de données d'étude. Le deuxième chapitre dresse un état de l'art des différents systèmes de gestion à même de répondre entièrement ou partiellement aux attentes des approches de type "grandes masses de données". Ce chapitre est enrichi par des illustrations provenant du jeu de données d'étude. Le troisième chapitre s'attache à comparer les grandes familles de systèmes NoSQL et *triplestore* sur la base des critères qui nous semblent pertinents à l'exemple du modèle de données, des mécanismes transactionnels et du passage à l'échelle. Un quatrième chapitre pose les bases de notre approche méthodologique, notamment en terme de définition d'un modèle de connaissances spatio-temporel qui tire parti de l'existant. Un cinquième chapitre ouvre quelques pistes sur les choix de systèmes de persistance à retenir. Enfin, le dernier chapitre permet de conclure et de dresser quelques perspectives de recherche.

CHAPITRE 1

CAS D'ÉTUDE AUTOUR DE LA CARTOGRAPHIE COLLABORATIVE

Nous avons définis une problématique de travail, de manière à pouvoir comparer par la suite les différentes catégories de systèmes de gestion de données étudiées. Les données portant sur les bâtiments d'enseignement et de recherche de l'Université Montpellier 2, sont exploitées pour illustrer les aspects relatifs notamment aux modèles de données et aux mécanismes d'accès aux données. L'intérêt est de pouvoir manipuler un jeu de données relativement restreint qui intègre une dimension spatiale et une dimension temporelle. Les bâtiments de l'UM2 sont localisés sur différents sites, à l'exemple du campus Triolet à Montpellier ou du site de l'IUT à Nîmes. Certains bâtiments ne sont gérés que depuis récemment par l'UM2 comme les différents bâtiments appartenant à la faculté d'éducation (nommée Espé).

1.1 Projet OpenStreetMap et son modèle de données

Le projet OpenStreetMap (OSM) [10] est un projet de cartographie collaborative qui a été lancé en 2004 en Angleterre pour, dans un premier temps, proposer une carte du Royaume Uni construite par tout un chacun, et accessible à tous. Fort de son succès, OSM est aujourd'hui un projet mondial et fédère l'activité de plus de 300 000 collaborateurs annuels pour l'édition de points d'intérêt (POI ou Point Of Interest) dans tous les pays du Monde. Les données du projet OSM sont rendues disponibles sous licence de type *creative commons* et différents outils sont également accessibles pour éditer, gérer, visualiser et traiter les collections de données spatialisées. OSM a défini un modèle semi-structuré au format XML qui offre la possibilité de représenter de manière simple et ouverte, des entités spatiales au travers des éléments `node`, `way` et `relation`. Ces éléments peuvent être enrichis par des étiquettes `tag` qui forment des couples propriété-valeur.

1.1.1 Grammaire et document OSM au format XML

Une portion de la grammaire (DTD pour Document Type Definition) d'OSM figure 1.1 . Un point d'intérêt nommé `node` possède différent attributs comme un identifiant `id`, une latitude `lat` et une longitude `lon`, une estampille temporelle `timestamp` et un identifiant d'utilisateur qui est à l'origine de sa création `uid`. Un `node` est une collection d'étiquettes `tag`. Une ligne ou un polygone sont définis au travers de la notion de `way` qui est une collection de `nd` (référence `node` et l'`id` de node) et de `tag` avec une possiblité de différents ordonnancements de ces éléments.

```
<!ELEMENT node (tag*)>
<!ATTLIST node id           CDATA #REQUIRED>
<!ATTLIST node lat          CDATA #REQUIRED>
<!ATTLIST node lon          CDATA #REQUIRED>
<!ATTLIST node changeset    CDATA #IMPLIED>
<!ATTLIST node visible      (true|false) #REQUIRED>
<!ATTLIST node user         CDATA #IMPLIED>
<!ATTLIST node timestamp    CDATA #IMPLIED>

<!ELEMENT way (tag*,nd,tag*,nd,(tag|nd)*)>
<!ATTLIST way id            CDATA #REQUIRED>
<!ATTLIST way changeset     CDATA #IMPLIED>
<!ATTLIST way visible       (true|false) #REQUIRED>
<!ATTLIST way user          CDATA #IMPLIED>
<!ATTLIST way timestamp     CDATA #IMPLIED>

<!ELEMENT tag EMPTY>
<!ATTLIST tag  k            CDATA #REQUIRED>
<!ATTLIST tag  v            CDATA #REQUIRED>

<!ELEMENT nd EMPTY>
<!ATTLIST nd ref            CDATA #REQUIRED>
```

FIGURE 1.1 – Une portion de la grammaire d'OSM

Quelques exemples de données au format XML, et conformes à la grammaire XML d'OpenStreetMap, sont fournis en guise d'illustration dans le figure1.3. L'objet de type `way` décrit, correspond au bâtiment administratif de l'UM2 (bâtiment 7) dans le figure 1.2. Deux nœuds constitutifs, parmi les sept décrits, de ce bâtiment sont également listés. Un troisième nœud qui correspond à la cabine téléphonique est également proposé en exemple.

FIGURE 1.2 – Visualisation du bâtiment 7 et du point téléphonique à l'entrée du bât

```
<node id="900514019" visible="true" version="2" changeset="7688411"
timestamp="2011-03-27T17:05:26Z" user="ReLuc" uid="35598" lat="43.6318636"
lon="3.8631036"/>

<node id="900441269" visible="true" version="2" changeset="7688411"
timestamp="2011-03-   27T17:05:26Z" user="ReLuc" uid="35598" lat="43.6313039"
lon="3.8634121"/>

<node id="2268519594" visible="true" version="2" changeset="15741320"
timestamp="2013-04-   15T19:36:17Z" user="Ptigrouick" uid="195175" lat="43.6316245"
lon="3.8632185">
  <tag k="amenity" v="telephone"/>
  <tag k="phone" v="0467610768"/>
</node>

<way id="76362815" visible="true" version="5" changeset="16507909"
 timestamp="2013-6-    11T09:58:21Z" user="ReLuc" uid="35598">
  <nd ref="900514019"/>
  <nd ref="900441269"/>
  <nd ref="900410570"/>
  <nd ref="900522843"/>
  <nd ref="900493513"/>
  <nd ref="900511879"/>
  <nd ref="900514019"/>
  <tag k="building" v="yes"/>
  <tag k="name" v="Bâtiment 7 Administration"/>
  <tag k="source" v="cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre.
    Mise à jour : 2010"/>
</way>
```

FIGURE 1.3 – Un petit jeu de données d'OSM

1.1.2 Rétroconception en modèles UML

Nous avons fait un travail de rétroconception pour en produire un modèle conceptuel sous la forme d'un diagramme de classe sous la notation UML (Figure 1.4).
Un diagramme d'objets est également proposé à partir des exemples de données provenant de la description des bâtiments du campus Triolet.

Diagramme de classes

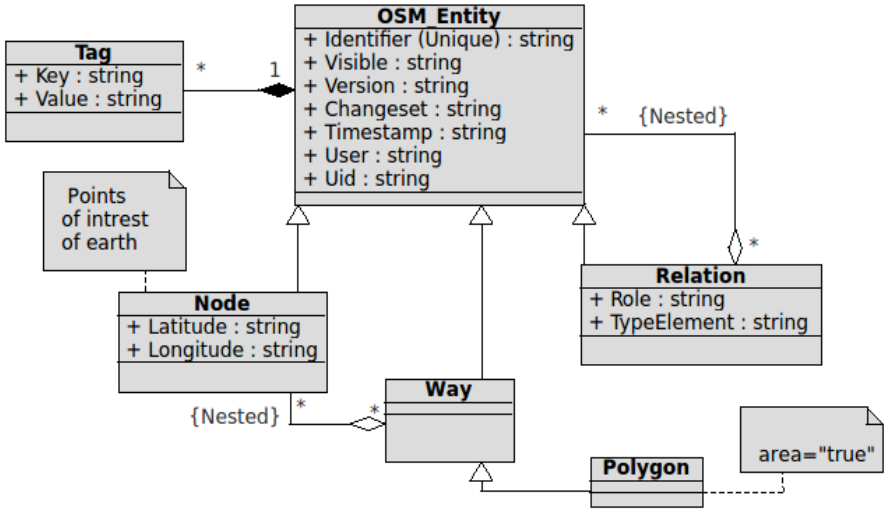

FIGURE 1.4 – Diagramme UML illustrant le modèle OpenStreetMap de manière partielle

Diagramme d'objets

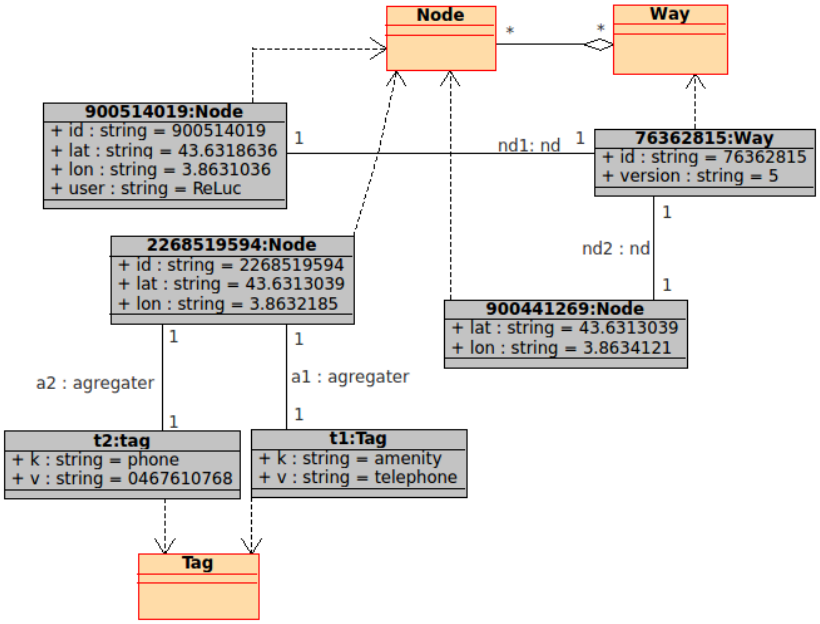

FIGURE 1.5 – Diagramme d'objets illustrant les données provenant du campus Triolet

1.2 Projet LinkedGeoData adossé à OSM

Le projet LinkedGeoData (LGDO) [5] présente plusieurs intérêts pour notre étude. Il exploite les données acquises au sein de l'initiative OpenStreetMap pour les rendre disponibles au format RDF. Ces objectifs sont d'ajouter au Web de données une dimension spatiale tout en respectant les principes clés de sources de données ouvertes. Ainsi, LGDO contribue à une utilisation ouverte de données décrivant des entités spatiales et permet d'associer à des données provenant d'autres initiations à l'exemple de DbPedia. LGDO va nous permettre de constituer le jeu de données RDF de nos exemples de *triplestores* à moindre effort.
L'exemple du bâtiment 7 a été obtenu par une interrogation sur le point d'accès de LGDO (`http://linkedgeodata.org/sparql`) et est repris ci-dessous en syntaxe RDF N3.

```
@prefix dc:     <http://purl.org/dc/elements/1.1/> .
@prefix lgdo:   <http://linkedgeodata.org/ontology/> .
@prefix lgdn:   <http://linkedgeodata.org/resource/node/> .
@prefix geo:    <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix lgdm:   <http://linkedgeodata.org/meta/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix spatial: <http://geovocab.org/spatial#> .
@prefix gadm-r: <http://linkedgeodata.org/ld/gadm2/resource/> .
@prefix gadm-o: <http://linkedgeodata.org/ld/gadm2/ontology/> .
@prefix geom:   <http://geovocab.org/geometry#> .
@prefix lgdw:   <http://linkedgeodata.org/resource/way/> .
@prefix lgdwn:  <http://linkedgeodata.org/resource/waynode/> .
@prefix meta:   <http://linkedgeodata.org/ld/meta/ontology/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix lgd:    <http://linkedgeodata.org/triplify/> .
@prefix ogc:    <http://www.opengis.net/ont/geosparql#> .
@prefix lgd-geom: <http://linkedgeodata.org/geometry/> .
@prefix dbpedia: <http://localhost:8080/resource/> .
@prefix owl:    <http://www.w3.org/2002/07/owl#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://linkedgeodata.org/data/triplify/way76362815?output=ttl>
    rdfs:label "RDF description of Bâtiment 7 Administration" ;
    foaf:primaryTopic lgd:way76362815 .

lgd:way7636281  a   lgdm:Way , spatial:Feature ;
    rdfs:label "Bâtiment 7 Administration" ;
    geom:geometry lgd-geom:way76362815 ;
    lgdo:building "true"^^xsd:boolean ;
    lgdo:changeset "16507909"^^xsd:int ;
    lgdo:source "cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre.
    Mise à jour : 2010" ;
    lgdo:version "5"^^xsd:int ;
    dcterms:contributor lgdo:user35598 ;
    dcterms:modified "2013-06-11T11:58:21"^^xsd:dateTime .
```

FIGURE 1.6 – Le projet LinkedGeoData adossé à OSM

CHAPITRE 2

ETAT DE L'ART

L'état de l'art présenté, aborde les grandes notions nécessaires à la compréhension de la suite de l'étude, en particulier pour ce qui concerne la comparaison des systèmes de gestion de données.

Dans la première partie de cette section, nous introduisons les notions à l'exemple de scalabilité ou de système distribué, qui sont ensuite exploitées dans notre comparatif à venir. Dans la seconde partie, nous revenons sur les systèmes de gestion de données les plus couramment exploités sur les dernières décennies. L'objectif est d'en dresser un panorama général sans considérer les nouveaux besoins (fortes volumétries de données distribuées et accessibilité à de nombreux usagers).

Nous résumons les grands principes de chaque catégorie de systèmes en détaillant leurs intérêts et les objectifs qu'ils cherchent à atteindre.

2.1 Notions préalables

Il nous semble utile de définir différents principes prépondérants dans le contexte des systèmes de gestion de données, qui reviendront souvent dans la suite de notre étude.

2.1.1 Modèles de données

Les données sont décrites, au sein d'une base de données, à l'aide d'un modèle, qui propose une organisation simplificatrice pour représenter ces données. La structure organisant les données doit être la plus fidèle possible au domaine modélisé. L'objectif d'un modèle est, en effet, de permettre de représenter les données du monde réel le plus simplement et le plus fidèlement possible. Des langages de définition et de manipulation des données viennent souvent outiller les modèles existants. Par exemple, le modèle relationnel propose d'organiser les données au travers de relations, qui d'un point de vue pratique, sont des tables qui vont pouvoir être définies et manipulées au travers du langage SQL (Structured Query Language). Les différents modèles de données diffèrent essentiellement par leur adéquation aux besoins des applications.
Dans le contexte de la modélisation de données spatio-temporelles, le choix du modèle de données est d'importance et doit aider l'utilisateur à mieux appréhender les données.[23]

2.1.2 Système distribué

Les bases de données distribuées ou réparties prennent en compte la nécessité des entreprises qui ont souvent, aujourd'hui, à gérer des informations provenant de plusieurs sites. L'idée est donc de ne pas centraliser l'information au niveau du siège central mais de laisser chaque site gérer sa propre information. Les données et les traitements sont disséminés sur plusieurs serveurs reliés par un réseau, mais des moyens sont mis en place pour que l'utilisateur perçoive la base de données comme une base unique. Les serveurs

hébergent chacun un SGBD [1] qui gère les données en local et participent aux transactions globales. Les données sont souvent répliquées sur plusieurs sites pour assurer une meilleure disponibilité des données. Le contrôle de la base reste centralisé, afin d'en garantir la cohérence.

L'intérêt de la répartition des données est multiple :

- gestion locale des données pour une meilleure adéquation aux besoins des entreprises
- répartition des données et des traitements sur les différents serveurs de manière à réduire les accès distribués et à équilibrer la charge du système (faciliter la montée en charge pendant les périodes d'accroissement de travail)
- disponibilité des données et résistance aux pannes
- extensibilité de l'architecture : un nouveau serveur va pouvoir s'ajouter sans créer de perturbation majeure.[23]

2.1.3 Passage à l'échelle ou scalabilité

La scalabilité désigne la capacité d'un système à s'adapter à un changement d'ordre de grandeur de la demande, soit parce que les usagers sont plus nombreux à solliciter le système au même moment, soit parce que les requêtes à satisfaire sont plus coûteuses. Un système se doit de supporter toute montée en charge, lors d'une surcharge de travail et il est attendu en particulier qu'il maintienne ses fonctionnalités et ses performances en cas de forte demande.[7] Si un serveur est doté de nouvelles ressources (mémoires vive ou secondaire, processeur, carte réseau en particulier) le rendant plus performant, alors il s'agit d'une *scalabilité verticale*. Si le système se voit ajouter de nouveaux serveurs, alors la scalabilité est dite *scalabilité horizontale*.

2.1.4 Théorème CAP (*Coherency*, *Availability*, *Partition Tolerance*)

Le théorème de Brewer ou CAP affirme qu'un système de calcul distribué ne peut respecter que deux contraintes sur les trois, énoncées ici :[13]

- **Cohérence** : tous les serveurs du système voient les mêmes données au même moment.
- **Disponibilité**(*Availability*) : garantit que toutes les requêtes reçoivent une réponse.
- **Résistance au morcellement**(*Partition Tolerance*) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (ou encore : en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome).

2.1.5 Mécanismes transactionnels

Une transaction est une unité de traitement constituée d'une séquence d'instructions SQL (actions de lecture et écriture sur les données de la base). Elle est vue comme un

1. Les bases de données sont gérées par des logiciels spécialisés appelés systèmes de gestion de bases de données (SGBD)

bloc et se termine soit par une validation explicite (*commit*) ou implicite (par exemple un ordre de création de table entraîne une validation de la transaction), soit par un abandon explicite (*rollback*) ou implicite lors de la survenue d'un problème. Le SGBD est un logiciel multi-utilisateurs et les mécanismes transactionnels vont être de nature à garantir la fiabilité de la base de données en cas d'accès concurrents et d'exécution simultanée de plusieurs transactions.[23]

Mécanismes ACID

Les modèles transactionnels vont garantir la cohérence des états successifs d'une base de données en faisant en sorte que les transactions respectent les propriétés dites ACID, c'est à dire d'atomicité, de cohérence, d'isolation et de durabilité.

- **Atomicité** : Une transaction suit une loi du tout ou rien, c'est à dire qu'elle doit s'effectuer entièrement ou pas du tout.
- **Cohérence** : Les contraintes d'intégrité qui s'appliquent sur les données de la base doivent être respectées dans tous les cas. En cas de problème, la transaction est annulée et le système revient à l'état cohérent précédent.
- **Isolation** : La transaction va s'exécuter dans un mode isolé et elle est la seule à voir les nouvelles valeurs des données qu'elle est en train de modifier. En attendant que cette transaction atteigne son point de validation (*commit*) ; le système garantit aux autres transactions, exécutées en parallèle sur le même système, l'accès aux anciennes valeurs des données en cours de modification.
- **Durabilité** : Après validation de la transaction, le système est dans un nouvel état stable et durable. Dans le cas contraire la transaction est annulée et le système retourne à l'état stable antérieur.

Mécanismes BASE (*Basically available*, *Soft state*, *Eventual consistency*)

Ils offrent une alternative moins contraignante que les mécanismes ACID. Dans le contexte de transactions distribuées, il devient par exemple difficile de poser l'ensemble des verrous nécessaires pour garantir que ces transactions soient isolées. Dans la vision BASE (nommé ainsi en référence au monde de la chimie), la disponibilité des données est privilégiée (*Basically available*) ; par contre, les contraintes portant sur l'état du système et la cohérence des données sont plus lâches. L'objectif est d'offrir des performances et une évolutivité accrues au système distribué.[13]

- **Disponibilité a priori** (*Basically available*) : le système garantit en tout premier, la disponibilité des données, en termes de ce qui est énoncé dans le théorème CAP.
- **État relâché** (*Soft state*) : l'état du système peut changer avec le temps, même sans écriture amenant à ce changement et peut se retrouver périodiquement dans un état instable. La stabilité du système est à atteindre dans le temps mais n'est pas requise à un instant précis.
- **Cohérence possible** (*Eventual consistency*) les données peuvent ne pas avoir la même valeur à un instant sur tous les serveurs du système. Là encore, il suffit que l'on converge vers cette uniformisation des valeurs dans le temps pour que la cohérence des données soit assurée.

Pour résumer et comparer les deux approches, un système assujetti aux mécanismes ACID, assure de ne délivrer que des données cohérentes alors qu'un système assujetti aux

mécanismes BASE assurer de délivrer les données, la plupart du temps cohérentes.

2.2 Logiciels de gestion de données

2.2.1 SGBDs relationnels

Un système de gestion de BD relationnel (SGBDR) est un logiciel qui permet de définir et d'exploiter des bases de données [2] (BD), structurées au travers du modèle relationnel. Un SGBDR va dans ce sens décliner trois fonctions principales :[26]

- **L'organisation des données au sein de relations**
  un SGBDR offre à l'utilisateur un langage dit de définition des données (LDD) permettant de décrire un schéma de BD. Le schéma relationnel comporte un ensemble de relations qui sont inter-reliées, l'utilisateur spécifie alors leurs propriétés, avec les attributs qui sont typés, et les contraintes d'intégrité à l'exemple des clés primaires, des clés étrangères et des contraintes de domaine. Le langage de définition de données est SQL (*Structured Query Language*) et est devenu un standard des organismes normalisateurs ANSI et ISO pour l'ensemble des SGBDR.[26]
- **La manipulation des données à l'aide d'un langage déclaratif**
  Le SGBDR permet de consulter et de mettre à jour les relations de la BD à l'aide de langages dits de manipulation de données (LMD). Il s'agit de langages déclaratifs avec lesquels on « déclare » ce que l'on veut obtenir de la BD. Le langage de manipulation de données est également SQL. Pour illustration, une requête SQL simple prend la forme :
  . `select` «attributs» `from` «relations» `where` «conditions».
  Ces trois clauses permettent d'exprimer les opérations spécifiques du relationnel comme la projection, la sélection ou la jointure. Le résultat d'une requête est une table temporaire constituée des attributs spécifiés dans la clause `select`.
- **L'administration des données**
  Cette fonction permet à un administrateur des données de faire évoluer la base de données dont il a la charge, tout au long de son cycle de vie, et veille à ce que toute évolution du schéma soit en accord avec les données contenues dans la base et les traitements qui peuvent en être fait. De même, chaque fois que l'intégrité des données peut être altérée par des requêtes de mise à jour ou encore par des accès malveillants, le SGBDR assure que les opérations s'exécutent en préservant la cohérence des données et soient conformes au schéma de la base de données. Pour tout ce qui touche au contrôle et à la supervision d'une base de données de manière à en garantir la fiabilité et la sécurité, des éléments de langage peuvent également venir faciliter le travail de l'administrateur de bases de données. Le langage SQL peut aussi être vu comme un langage de contrôle de données (LCD). Certains SGBDR comme Oracle ou PostgreSQL ont défini des langages procéduraux et propriétaires venant compléter le langage SQL pour renforcer les aspects liés à la sécurité et à la supervision au sein des systèmes.

2. Une base de données est une collection de données structurées mémorisées sur un support permanent, qui vont pouvoir être restituées de manière efficace à l'usager

Ainsi le langage PL/SQL est adapté à la définition de déclencheurs (*triggers*) qui vont contrôler l'activité des usagers et prévenir les opérations illicites au sein de la base de données.
Le langage SQL est un des atouts majeurs des SGBDR. Il fait office de langage de définition, de manipulation et de contrôle de données. Il est devenu au travers du temps un standard, et a été soumis à différentes évolutions (la plus récente est SQL3) qui en ont fait un langage expressif et efficace. Le simple fait de pouvoir changer de SGBDR sans pour autant changer de langage, est un véritable confort de travail pour les gestionnaires de bases de données comme pour les administrateurs de bases de données.

- **Le modèle relationnel** est de loin le modèle le plus exploité depuis sa définition au début des années 1970 [11], il doit son succès à la relative simplicité de son concept central de relation et à son adossement aux mathématiques et à la théorie des ensembles en particulier. Un modèle relationnel est constitué de relations (des tables) qui apparaissent comme une des listes non ordonnées d'attributs (les colonnes) et dans lesquelles viennent se ranger des n-uplets (les lignes). Chaque n-uplet décrit une collection de données de façon unique. Une relation organise ainsi une collection de données ayant la même structure. Outre l'efficacité de son modèle organisationnel, l'intérêt du modèle relationnel provient également de :[23]
  - L'existence de l'algèbre relationnelle qui permet de manipuler et de combiner les relations de manière abstraite et qui trouve un prolongement direct dans la syntaxe du langage SQL.
  - La théorie de la normalisation qui permet de garantir la qualité et l'efficacité des modèles relationnels construits en s'assurant de la cohérence et de la non duplication des données. Les trois premières formes normales sont basées sur la notion de dépendances fonctionnelles

2.2.2 Systèmes NoSQL

Les systèmes non-relationnels NoSQL, pour Not Only SQL, ont émergé récemment. Ils apportent des solutions de stockage de données complémentaires au relationnel. Ces systèmes présentent généralement six caractéristiques principales qui sont résumées ici :[7]

- Capacité au passage à l'échelle horizontal dans le cadre d'opérations de lecture/écriture simples. La montée en charge pour ce qui relève du traitement des transactions va être répartie, de manière équilibrée, sur plusieurs serveurs et les temps de réponse vont rester corrects pour les différents usagers.
- Capacité à partitionner et à répliquer les données sur plusieurs serveurs. Des politiques d'allocation de blocs de données sont définies.
- Simplicité des accès aux données. Les mécanismes d'appel au système s'appuient sur des interfaces de programmation ou des protocoles d'accès (contrairement aux liaisons exploitant le langage SQL dans le contexte des SGBD relationnels)
- Modèle de concurrence moins contraint que le modèle transactionnel ACID des SGBD relationnels. La cohérence sur les données qui impose que les effets d'une opération d'écriture soit observés sur tous les répliques distribués sur les différents serveurs, n'est pas toujours respectée.
- Utilisation efficace d'index distribués et de structures de mémoire cache.

Certains systèmes exploitent directement la mémoire vive pour le stockage des données, avec la possibilité de sauvegarde d'instantanés et de répliquer les données en mémoire secondaire, alors que d'autres sont conçus pour le stockage sur disque avec une possibilité de mémoire cache en mémoire RAM.
- Modèle de données ouvert. L'importance est donnée à l'extensibilité du modèle.
- **Mécanisme Map/Reduce** : L'un des intérêts de la mouvance NoSQL est d'intégrer le paradigme dit : *Map/Reduce* qui consiste à partitionner les données afin de les traiter en parallèle. Le principe est de diviser les données à traiter en partitions indépendantes, traiter ces partitions en parallèle et finalement de combiner les résultats. Cette technique se décompose en deux grandes étapes :
  - Étape **Map** consiste en l'étape de découpage puis de distribution des différentes partitions de données constituées. Un item analyse un problème, le découpe en sous-problèmes qu'il délègue à d'autre items (récursivement). Chaque item d'une liste d'objets clé/valeur est passé à la fonction map qui va retourner un nouvel élément clé/valeur.
  - Étape **Reduce** est l'opération inverse du Map qui consiste à récolter tous les résultats calculés en parallèle et les fusionner (*reduce*) en un seul résultat global. Cette opération permet d'appliquer une opération sur la liste des items.

Les bases de données NoSQL génèrent intérêt et critique. Intérêt car elles répondent à des exigences qui sont très importantes dans la mise en œuvre d'applications à grande échelle. Critique en raison de la comparaison avec les réalisations relationnelles connues. L'un des problèmes majeurs souvent mentionné est l'hétérogénéité des languages de programmation offerts aux développeurs et aux utilisateurs pour accéder et manipuler les données. Cela nécessite par exemple des efforts significatifs pour faire migrer une source de données d'un système à l'autre. La plupart des systèmes NoSQL ont été développés indépendamment les uns des autres, et sont classés en quatre grandes familles :[4, 7]

Les systèmes clé-valeur

Le modèle sous-jacent peut être assimilé à un tableau associatif distribué. Les données sont donc simplement représentées sous la forme de couples clé/valeur. La base de données repose essentiellement sur une structure de type *hashmap* qui contient des objets (appelés valeurs), chaque objet étant identifié par une clé unique.
Les systèmes existants diffèrent par les types des valeurs et des clés. Ainsi, en ce qui concerne les clés, certains systèmes les considèrent comme des chaînes de caractères, tandis que d'autres permettent une structure hiérarchique composée de plusieurs parties. Les valeurs peuvent être des éléments simples tels que des chaînes de caractères, des nombres entiers, ou des objets structurés.

L'absence de structure élaborée a un impact important sur la manière de requêter les données stockées. En effet, toute la puissance expressive pouvant être exploitée habituellement directement dans les requêtes SQL, devra être portée par l'applicatif qui interroge la source de données. Néanmoins, la communication avec la base de données, se résume en général aux opérations élémentaires de type lecture/écriture sur une donnée désignées par `PUT`, `GET` et `DELETE`. Les solutions technologiques les plus connues sont *Redis*, *Riak* et *Voldemort* (créé par *LinkedIn*) et *Scalaris*.

Les systèmes orientés document

Le paradigme des systèmes orientés document est également un paradigme clé-valeur. La valeur est cependant dotée d'une organisation plus aboutie que dans le cas des systèmes clé-valeur ; qui prend la forme d'un document défini dans les langages de description JSON (*JavaScript Object Notation*) ou XML. Chaque document est composé d'un ensemble de champs (qui peuvent à leur tour être composés d'éléments imbriqués) et est associé à un identificateur unique, à des fins d'indexation et de restitution. En général, ces systèmes offrent un langage de requête plus riche que ceux des autres catégories NoSQL, qui peut exploiter la structure des objets qu'ils stockent. L'avantage est de pouvoir retourner, sur la base d'un appel sur une simple valeur de clé, un ensemble d'informations structurées de manière hiérarchique. La même opération de consultation impliquerait une requête avec plusieurs jointures dans le monde du relationnel.
Pour ce modèle, les implémentations les plus populaires sont *CouchDB d'Apache*, *SimpleDB*, *Terrastore* et *MongoDB*.

La figure 2.1 reprend le jeu de données OSM. Une collection d'éléments de type `Node` (appelée *Nodes*) va permettre de structurer l'ensemble des points d'intérêt géographiques au sein d'un même document. Les propriétés élémentaires comme `Id`, `Version`,. . .peuvent être représentées par un couple propriété - valeur littérale alors que la collection de tags décrivant de manière multiple un élément est représentée par un couple propriété nommé *tags* - valeur de type tableau (matérialisée par des crochets).

```
nodes: [
        1001: {
             id: "656934",
             visible: "true",
             verrsion :"15",
             changeset: "6810331"
             timestamp: "2010-1230T15:49:07Z",
             user: "ReLuc",
             uid: "35598",
             Lat: "43.6295863",
             lon: "3.8700938",
             tags: [
                {
                   k: "name",
                   v: "Pharmacie vertBois",
                },
                {
                   k: "source",
                   v: "Montpellier",
                }
            ]
        },
        1002 : {
              id: "20933254",
              ...
        },
```

FIGURE 2.1 – Une collection Nodes dans MongoDB

Les systèmes orientés colonne

Le modèle sous-jacent s'appuie sur la notion de table et peut faire penser au modèle relationnel. La structure tabulaire est cependant beaucoup plus flexible, et comprend des familles de colonnes qui vont agréger des colonnes qui ne sont pas prédéfinies. Le nombre de colonnes dans chaque famille est dynamique. A la différence, dans une table du modèle relationnel, le nombre de colonnes est fixé dès la création du schéma de la table et ce nombre reste le même pour tous les tuples de cette table. Un intérêt du système à colonnes est de permettre que le nombre de colonnes puisse varier d'un enregistrement à un autre et ainsi éviter de gérer des valeurs d'attributs non renseignées (`null` dans le langage SQL) pour certains tuples. Les colonnes sont regroupées en familles ce qui facilite le partitionnement vertical et simplifie l'organisation des données au niveau physique.
Les solutions existantes les plus connues se nomment HBase (implémentation Open Source du modèle *BigTable* publié par *Google*) ainsi que *Cassandra* (projet Apache qui respecte l'architecture distribuée de *Dynamo d'Amazon* et le modèle *BigTable* de *Google*).
La figure 2.2 donne un exemple de l'organisation des données provenant d'OSM au sein du système HBASE. Nous avons défini une table nommée `Elements` qui va regrouper tous les enregistrements décrivant des entités spatiales (`Node`, `Way` et `Relation`). Les familles de colonnes jouent un rôle important et sont également fixées à la création du schéma. Nous spécifions la famille de colonnes nommée `CommunsInfo` qui pose un cadre pour toutes les informations générales décrivant les entités spatiales. Nous spécifions également la famille de colonnes `TagsInfo` qui va regrouper l'ensemble des paires propriété-valeur désigné par `Tag` dans OSM. HBASE est particulièrement adapté à ce genre d'organisation générique. Une colonne est créée à la volée quand un nouveau `Tag` vient décrire une entité, à l'exemple de la propriété `Amenity` qui prend la valeur `University`. La famille de colonnes `OrganisationInfo` permet d'organiser les attributs de géoréférencement comme `latitude` et `longitude` d'un élément de type `Node` et les attributs décrivant chaque nœud constitutif pour un élément de type `Way`.

Les systèmes orientés graphe

Les systèmes orientés graphe (GOOD) sont radicalement différents des trois catégories de systèmes NoSQL que nous venons de présenter. Alors que les systèmes clé-valeur, orientés document et orienté colonne s'attachent à proposer une représentation qui agrège les données ; les systèmes orientés graphe vont plutôt s'inspirer des systèmes orientés objet et des anciens systèmes hiérarchiques. Les objets vont être les nœuds du système et les liens qui se nouent entre les objets vont en être les arcs. Les nœuds et les arcs vont pouvoir être caractérisés par des propriétés. Les systèmes orientés graphe se basent pour une majorité d'entre eux sur les principes des graphes attribués. Le modèle orienté graphe va ainsi pouvoir bénéficier de toutes les avancées autour de la théorie des graphes sur les dernières décennies. De même la représentation de l'information sous forme de graphe est à la fois intuitive et visuelle, ce qui va en faciliter l'appropriation par les usagers finaux. Les systèmes orientés graphe sont particulièrement adaptés à la gestion de réseaux sociaux, de voies de communication (réseaux ferroviaires, routiers, aériens, ...), ou encore de voies métaboliques. Ces systèmes vont donc se révéler des candidats intéressants pour la représentation d'entités spatialisées et des relations qui se nouent entre elles. Différentes solutions sont disponibles à l'exemple de *Neo4J*, *FlockDB*

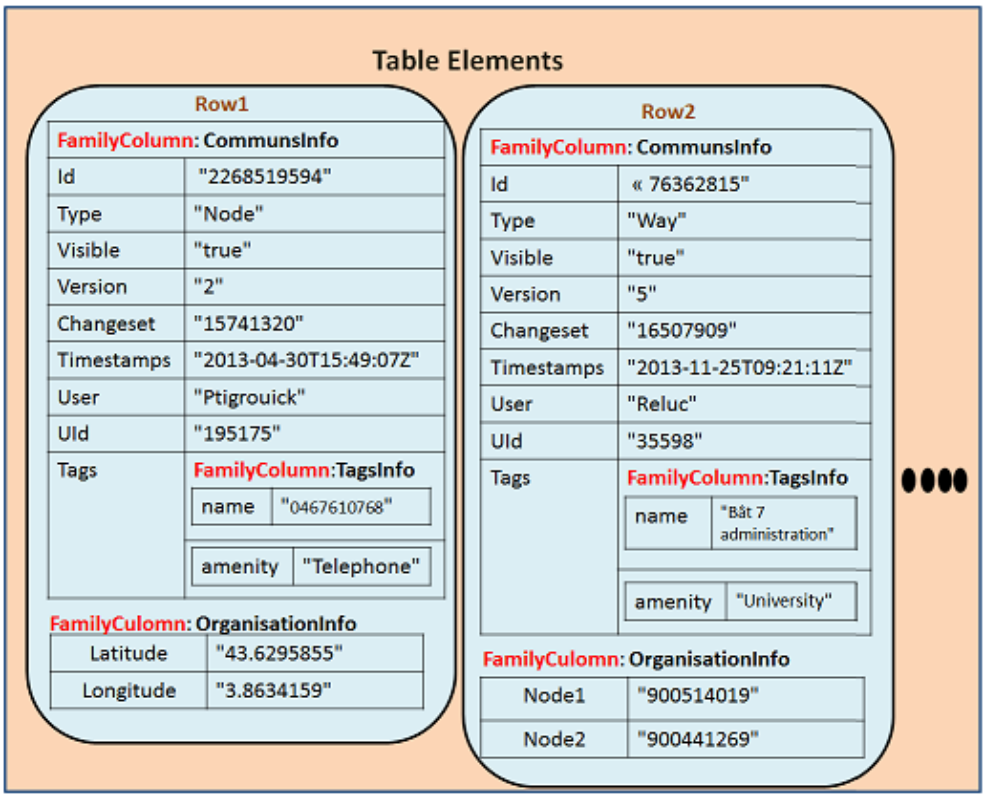

FIGURE 2.2 – Un exemple de table et de deux tuples dans HBase

ou *OrientDB*. *Neo4J* possède une extension pour le spatial qui lui permet de se positionner en tant que SIG (Système d'information géographique). Les systèmes de gestion de triplets au format RDF que nous abordons dans la sous-section suivante, pourraient également se ranger dans les systèmes orientés graphes.

2.2.3 Systèmes de gestion de triplets RDF ou triplestores

De plus en plus de jeux de données sont rendus disponibles sous la forme de fichiers au format RDF. Il semble logique de considérer le stockage et la consultation de ces gros volumes de données sans changer de modèle de représentation. Le langage de requête SPARQL (*SPARQL Protocol and RDF Query Language*) [3], développé pour les besoins du Web de données, peut être exploité directement pour tout ce qui concerne la consultation ou la manipulation de triplets RDF, rendus persistants par ailleurs.

Nous introduisons rapidement dans un premier temps les langages RDF, RDFS, OWL et SPARQL avant d'aborder véritablement les triplestores.

Les langages du Web de données

RDF (*Ressource description framework*) est un langage défini à l'origine pour décrire tout type de ressource de manière décentralisée au travers de couples propriété/valeur.

3. SPARQL est aussi un protocole d'accès aux données RDF

L'entité centrale dans le modèle RDF est le triplet (encore appelé *déclaration* ou *statement*). Elle résulte d'une association de trois éléments nommés **sujet**, **prédicat** ou propriété et **objet** : [12]

- **Le sujet** représente la ressource à décrire et est soit identifié par une IRI (Identificateur de ressource internationalisé), soit défini comme un nœud (ou ressource) anonyme.
- **Le prédicat** représente un type de propriété applicable à cette ressource
- **L'objet** représente la valeur prise par la propriété pour le sujet considéré. L'objet peut se révéler être une valeur terminale (littérale), une ressource étiquetée (identifiée par une IRI) ou anonyme.

Un modèle RDF va être un graphe qui contient l'ensemble des triplets inter-reliés.
Les langages RDFS (*RDF Schema*) et OWL (*Ontology Web Language*) viennent enrichir le langage RDF en proposant des éléments de modélisation supplémentaires, à l'exemple des hiérarchies de classes et de propriétés pour RDFS ou de la notion de classes disjointes ou complémentaires pour OWL.

Les grands principes des triplestores

L'idée est de gérer de la manière la plus efficace possible les triplets d'un ou de plusieurs graphes RDF. La notion de quadruplet est ainsi mise en avant, l'information sur l'espace de noms du graphe (sous la forme d'une IRI) est ajoutée dans une quatrième colonne à chaque triplet stocké. Plusieurs organisations de l'information sont possibles. L'organisation la plus simple va consister en une relation d'arité 4 (une table à quatre colonnes) dont les tuples vont être les quadruplets avec par exemple dans l'ordre : le sujet, le prédicat, l'objet et enfin l'IRI désignant le modèle RDF d'appartenance. Les systèmes de gestion de triplets RDF sont nommés sous leur vocable anglais : triplestore. Les triplestores sont parfois désignés comme des systèmes de gestion de données **sans schéma (*schemaless*)**. Plus exactement, l'organisation des structures contenant les triplets est fixe quel que soit le domaine décrit. Il s'agira toujours de ressources représentées au travers de différentes propriétés qui pointeront sur des valeurs terminales ou qui mettront les ressources en relation. L'importance est de définir une structure adaptée à l'organisation de triplets ou de quadruplets. La spécificité de chaque modèle est explicitée au travers des types de ressources et de propriétés décrites.

Nous distinguons trois catégories de triplestores qui diffèrent par leurs architectures :[16, 24] les triplestores natifs, les triplestores adossés aux SGBD relationnels et les triplestores qui fonctionnent sur la base de transformateurs (*wrappers*) afin d'amener les sources de données existantes (souvent au format relationnel) jusqu'à des graphes RDF.

Triplestores natifs : Un moteur de base de données est défini entièrement et est optimisé pour le traitement de triplets RDF, et ce indépendamment de tout autre système de gestion de base de données (SGBD). Les données sont stockées directement dans un système de fichiers, soit dans un fichier centralisé unique ou un bien dans un fichier réparti sur plusieurs serveurs. Pour exemple dans cette catégorie nous pouvons citer les systèmes *AllegroGraph*, *Jena TDB*, *Kowari*, *OWLIM* et *ClioPatria*.

Les SGBDs adossés au relationnel (*DBMS-backed Stores*) : Ces systèmes utilisent les architectures fonctionnelles de systèmes de gestion de base de données relationnels existants à l'exemple de PostgreSQL, Oracle ou MySQL. Différents modèles organisationnels permettent de rendre compte des modèles RDF au travers de schémas relationnels sous-jacents. Ces modèles sont dits génériques ou *oblivious* lorsqu'ils ne cherchent pas à restranscrire la connaissance du modèle RDF. Il va s'agir d'une relation d'arité 3 ou 4 dans laquelle tous les triplets vont être stockés. Ces modèles sont dits *aware* lorsqu'ils s'appuient sur différentes relations qui retranscrivent par exemple les ressources qui sont souvent les sujets des triplets ou bien les propriétés qui mettent en relation des ressources entre elles. Jena SDB, YARS2, 3store, ARC, Semantics Platform sont quelques exemples de triplestores adossés au relationnel.

Les triplestores hybrides prennent en charge les deux orientations architecturales (natives et adossées aux SGBDR). Sesame et Virtuoso en sont des exemples.[**?**]

Nous illustrons l'organisation des modèles RDF qui seront pris en charge au sein des triplestores. Les documents RDF peuvent être écrits en différentes syntaxes, y compris en XML (RDF/XML). Mais RDF en soi n'est pas un dialecte XML. Il est possible d'avoir recours à d'autres syntaxes pour exprimer les triplets à l'exemple de N3. Le schéma 2.3 est une visualisation du graphe de triplets dont la présentation XML est donnée en figure 1.6. Nous reprenons intégralement la structuration proposée par le projet LGDO. La visualisation s'attache à décrire le nœud *lgd :way76362815* au travers de ses relations avec d'autres ressources.

- **lgd :way76362815** est ce qu'on appelle une ressource, ou encore un sujet.
- L'arc **rdfs :label** est ce qu'on appelle un prédicat ou encore une propriété.
- **batiment7 Administration** est ce qu'on appelle une valeur ou encore un objet, une cible (target) qui peut être aussi une ressource par exemple **lgdo :user35598**.

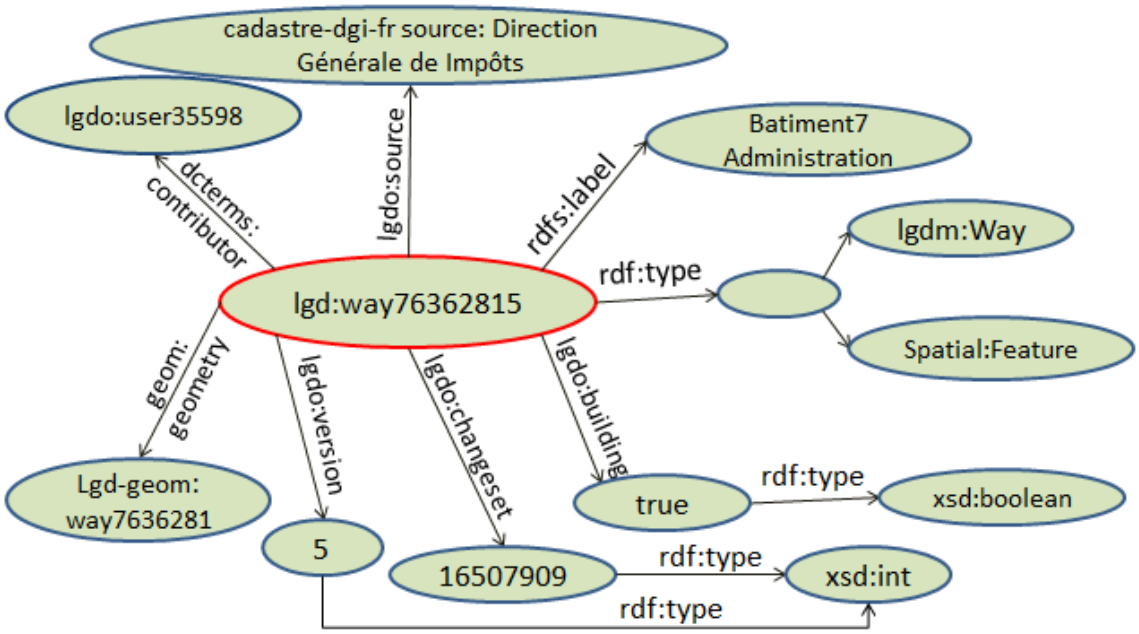

FIGURE 2.3 – Visualisation du modèle RDF pour l'entité correspondant au bâtiment 7

Les wrappers RDF : Ceux sont des composants logiciels dits légers mis en place au dessus d'une source de données existante et qui permet la publication des données qui y sont stockées au format RDF, sans en affecter les infrastructures existantes de stockage. Des wrappers RDF sont rendus disponibles pour des bases de données relationnelles (D2RQ Server3, Triplify), des protocoles tels que OAI-PMH (OAI2LOD Server), des formats de fichiers structurés (TripCel), des systèmes de gestion de fichiers complets (TripFS) et d'autres types de sources de données. Selon les caractéristiques de la source de données sous-jacente, les wrappers peuvent également fournir un accès via le langage de requête SPARQL. Les wrappers RDF ne rendent en général des services d'accès aux données qu'en lecture seule. Ils offrent donc moins de fonctionnalités que les deux premiers types de triplestores qui offrent toutes les fonctionnalités de mise à jour des données.

CHAPITRE 3

COMPARAISON DES DIFFÉRENTS SYSTÈMES DE GESTION DE DONNÉES

Nous avons pu montrer, dans l'état de l'art, que de nombreux choix et orientations étaient rendus possibles pour ce qui concerne la gestion de données. Les dernières avancées technologiques, notamment autour du Web, alliées aux connaissances théoriques acquises dans le contexte des bases de données, ont contribué à faire émerger de nouvelles solutions de stockage de l'information. Les systèmes se doivent d'être décentralisés, accessibles à de larges communautés d'usagers et avoir la capacité de gérer efficacement de grandes masses de données. Nous nous efforçons, dans cette nouvelle section, de dresser un comparatif de ces différents systèmes, en gardant à l'esprit ces derniers paramètres. Nous y ajoutons également les considérations autour de la spécificité de la gestion de données spatio-temporelles.

Les catégories de système introduites dans l'état de l'art, sont discutées sur leurs capacités à proposer des mécanismes adaptés aux orientations énumérées ici :

- passage à l'échelle et notamment scalabilité horizontale,
- partitionnement des données
- réplication des données
- mécanismes transactionnels
- expressivité et flexibilité des modèles de données
- accès aux données et langages de requêtage

3.1 SGBDs Relationnels

3.1.1 Forces

Les SGBDR sont les principaux acteurs du marché depuis plus de trente ans et fournissent un ensemble intégré de fonctionnalités qui vont apporter des réponses robustes et matures face aux demandes diverses de stockage et de traitement de l'information. Les qualités des SGBDR portent notamment sur les mécanismes mis en jeu pour organiser et manipuler efficacement et de manière sécurisée des données fiables. Les usagers sont identifiés via des profils et des rôles. Les bases de données sont maintenues dans des états stables et cohérents et les mécanismes transactionnels arbitrent les accès concurrents. Le modèle relationnel permet d'organiser au mieux les données lorsqu'elles ne se révèlent pas trop complexes ou trop évolutives. L'emprunt à la théorie des ensembles et la possibilité d'exprimer un grand nombre de requêtes au travers de l'algèbre relationnelle apporte de la rigueur à la manière de consulter les données. La théorie de la normalisation apporte de la rigueur à la manière de construire les schémas relationnels. Le partage du standard SQL par l'ensemble des SGBDR facilitent à la fois les migrations d'un système à un autre et les échanges de données d'un système à un autre. De même les principes d'imbrication de requêtes SQL dans un langage de programmation hôte sont partageables par l'en-

semble des SGBDR. Ainsi, la couche Java de connectivité aux bases de données JDBC peut être en grande partie réutilisable d'un SGBD à l'autre, seuls les aspects purement techniques de connexion sont spécifiques à chaque système. Un développeur va, de ce fait, pouvoir définir des couches de traitements pouvant s'adapter à différents SGBDR et peut également accéder à l'information contenue dans différents systèmes à partir de la même couche applicative. Des surcouches procédurales (PL/SQL pour Oracle et PgPLSQL pour PostgreSQL) ont permis pour les SGBDR les plus évolués de renforcer la sécurité par le biais de déclencheurs, d'internaliser certains traitements ou encore d'intégrer la vision relationnelle-objet au travers de la notion de type de données abstrait (ADT et standard SQL3).[11]

Si nous nous intéressons à l'information géographique, différentes surcouches sont disponibles pour gérer la géométrie des entités spatiales. Par exemple, PostgreSQL propose une extension PostGIS qui est construite à partir de fonctions définies dans le langage procédural PgPLSQL. Oracle propose l'extension SDO (Spatial Data Object) qui est construite à partir de fonctions définies dans le langage procédural PL/SQL.

3.1.2 Faiblesses

Certaines limites du relationnel sont connues et admises depuis de nombreuses années. Ces limites sont contrebalancées par de multiples avantages et ne remettent pas en cause le choix du relationnel dans une grande majorité des cas. Ces premières limites sont reprises ici :

- Le modèle relationnel n'est pas le mieux adapté à la représentation d'entités complexes. Ainsi, l'expression de liens d'héritage et d'agrégation est rendue difficile, notamment en raison de la contrainte imposée par la première forme normale (atomicité de la valeur prise par les attributs). Pour illustration, le simple fait de gérer une base de données qui va porter sur des bâtiments typés en bâtiments de recherche, d'enseignement ou administratif peut amener à définir quatre relations (bâtiment, bâtimentRecherche, bâtimentEnseignement et bâtimentAdministratif). L'information associée à un bâtiment en particulier va être décomposée et organisée au sein de deux de ces relations en fonction du type du bâtiment.
- Le modèle relationnel n'est pas non plus le mieux adapté à faire face à de l'évolution de schéma. Ainsi, la structure d'une relation ne peut être amenée à être largement révisée. Un attribut peut être ajouté ou modifié à la marge. Pour illustration, il est difficile de prendre en charge au sein d'une même relation Bâtiment, des tuples associés à des bâtiments qui sont décrits par de multiples caractéristiques qui ne sont pas forcément les mêmes.
- La distorsion des langages (ou *impedance mismatch*) : il est difficile de mettre en correspondance les données provenant d'un schéma relationnel avec les données manipulées à l'aide d'un langage impératif ou objet. Les mécanismes d'accès aux données dans un schéma relationnel s'appuient sur la valeur unique et toujours renseignée de la clé primaire, alors que par exemple dans un modèle objet, les accès aux données s'effectuent via des pointeurs gérés par le système. Cette non-concordance nécessite des applicatifs de mise en correspondance (*mapping*) à l'exemple d'*Hibernate* dans le contexte du *mapping* objet-relationnel.

Les limites des systèmes relationnels, qui ont conduit à l'apparition de la mouvance NoSQL, sont à mettre sur le compte de la difficulté de ces systèmes à fonctionner au travers d'architectures distribuées ouvertes à de nombreux utilisateurs. Les fournisseurs de services en ligne (Yahoo, Google) ou plus récemment les acteurs du web social (*Facebook*, *Twitter* ou *LinkedIn*) se sont trouvés les premiers confrontés à la nécessité de rendre disponibles de grands volumes de données à une large communauté d'usagers répartis à travers le monde.[7, 4]

- les systèmes relationnels ont été pensés pour fonctionner dans le contexte d'architectures centralisées. Différents réflexions ont été menées de manière à proposer un partitionnement des schémas relationnels et d'allouer ces portions de schéma à différents serveurs. Ainsi, il est envisageable de fragmenter un schéma horizontalement (les tuples d'une relation sont partitionnés), verticalement (les colonnes d'une relation sont partitionnées) ou bien les deux. Les fonctionnalités DbLink de Oracle permettent ainsi d'envisager l'implantation de serveurs répartis. Ces solutions restent cependant exploitées à l'échelle de la distribution sur quelques nœuds serveurs mais ne semblent pas pouvoir répondre à des besoins plus vastes et plus ouverts.
- l'observation stricte des contraintes d'atomicité, d'isolation, de cohérence et de durabilité, alourdit l'exécution de transactions distribuées. Une transaction distribuée est composée de plusieurs transactions dites locales qui s'exécutent sur différents sites. Pour garantir le respect des contraintes ACID, il faut se doter de systèmes supplémentaires chargés de coordonner les transactions globales et de s'appuyer sur de nouveaux mécanismes de validation dit 2PC (validation à deux phases).

En résumé, les SGDB relationnels restent toujours aussi performants dans une vision centralisée et classique de la gestion de données. De nouveaux besoins qui sont liés en particulier à la mise en place de services sur le web axés sur une utilisation différente de l'information ont conduit à envisager de nouvelles approches.

3.2 SGBD NoSQL

Les systèmes NoSQL ont été conçus de manière indépendante, avec chacun des objectifs spécifiques ; l'objectif général est cependant d'offrir aux usagers la possibilité de réaliser des opérations de lecture/écriture élémentaires sur des structures de données qui peuvent être changeantes. En pratique, cette simplicité d'exploitation va faciliter la possibilité de faire évoluer à la demande les modèles et de traiter massivement de larges volumes de données. Une caractéristique commune à presque tous ces systèmes, porte sur la gestion individualisée d'éléments, identifiés à l'aide de clés uniques. Les systèmes diffèrent sur les types de constructeurs (`map`, `set`, *list*) qui peuvent être exploités pour la définition des éléments, et en ce sens sur la possibilité d'agréger ces éléments sur différents niveaux. Ainsi, les structures sont dites flexibles, en ce sens que les schémas de données sont rendus ouverts. La littérature parle de systèmes sans schéma (*schemaless*), ce qui est un peu exagéré puisque les schémas sont génériques et pré-existent.[4]

La notion de paires attribut-valeur est essentielle et va permettre de caractériser toute ressource à décrire. Les structures de données, mises à contribution pour structurer

les paires attribut-valeur sont en particulier : [7]

- Un **tuple** a déjà été abordé au sein des systèmes relationnels. Les noms d'attributs sont prédéfinis dans un schéma, et leurs valeurs doivent être scalaires. Les valeurs sont référencées par le nom d'attribut, alors que dans une liste ou dans un tableau ou une liste, elles sont référencées par leur position ordinale.
- Un **document** permet aux valeurs de se décliner en documents imbriqués, listes ou valeurs scalaires. Un document diffère d'un tuple en ce que les attributs ne sont pas définis dans un schéma global, et de nouvelles valeurs peuvent être ajoutées dynamiquement.
- Un **enregistrement extensible** est une structure hybride entre le tuple et le document. Des familles d'attributs sont définis dans un schéma, mais de nouveaux attributs peuvent être ajoutés à la volée (au sein d'une famille d'attribut)
- Un **objet** est analogue à un objet dans les langages de programmation, mais sans les méthodes. Les valeurs peuvent être des références ou bien des objets imbriqués.

Par la suite nous présentons une classification des différents systèmes sur les fonctionnalités de modélisation de données et de leur modèle de requête.

3.2.1 Paradigme clé/valeur

Dans cette approche, chaque objet est doté d'une clé unique qui constitue la seule manière d'y accéder. Ces systèmes sont fortement scalables. Des mécanismes de distribution des objets sur la base de la valeur de leur clé sont disponibles. Ces systèmes offrent également une grande latitude dans l'évolution. La structure de l'objet est en effet libre et le plus souvent laissé à la charge du développeur de l'application (XML, JSON, ...). Elle peut porter sur des types simples tels que les chaînes de caractères et les nombres entiers, ou bien des types complexes, en fonction de la nature des données à gérer. En ce qui concerne les clés, elles peuvent être de simples chaînes de caractères ou bien relever de structures hiérarchiques.[4]

Ces systèmes ne maintiennent que les quatre opérations élémentaires CRUD `Create, Read, Update, Delete`, habituelles :

- **Create** : créer un nouvel objet avec clé et valeur $\longrightarrow$ create(key, value).
- **Read** : lit un objet à partir de sa clé $\longrightarrow$ read(key).
- **Update** : met à jour la valeur d'un objet à partir de sa clé $\longrightarrow$ update(key, value).
- **Delete** : supprime un objet à partir de sa clé $\longrightarrow$ delete(key).

Sur le plan fonctionnel, les services de structuration et d'accès très limités de ces systèmes, leur autorisent des performances élevées en lecture et en écriture ainsi qu'une scalabilité horizontale pouvant aller jusqu'à plusieurs centaines de serveurs. Le besoin en terme de scalabilité verticale est rendu moins nécessaire de par le caractère très simple des opérations effectuées.

Ces systèmes sont généralement de bonnes solutions quand les besoins en terme de gestion de données sont très simples et que l'application construite est centrée sur un seul type d'objet. Pour illustration, ces systèmes sont exploités dans le contexte de mises en place d'annuaires de personnels au sein d'entreprises, de système de stockage de cache ou de sessions distribuées. Si les besoins applicatifs nécessitent plus d'expressivité, il est

préférable si nous voulons rester dans un modèles de données apparenté, d'utiliser une base de données orientée document.

La figure 3.1 donne quelques exemples des systèmes clé-valeur et compare leur caractéristiques qui importent dans notre étude.

Système	Réplication	Modèle Transactionnel	Passage à l'échelle	Partitionnement automatique	Control concurrence	langage de requêtes
Voldemort	Symétrique Async	Théorème CAP AP	Horizontal, liner (Read/write)	Oui	MVCC	APIcall
Riak	Multi-Master Async	BASE Théorème CAP(AP)	Horizontal	Oui Data Type : Binaire	MVCC	JavaScript Erlang REST
Redis	Master-slave Async	Théorème CAP	Horizontal	Non	Locks	Lua APIcall
Scalaris	Multi-Master Sync	ACID	Horizontal	Oui	Locks	JSON

FIGURE 3.1 – Comparaison des systèmes clé-valeur NoSQL

Voldemort est un système développé par LinkedIn à des finalités d'usage interne puis rendu disponible en open source. Un travail important a été mené autour de l'optimisation de la communication entre les différents nœuds du système.

Riak est l'implémentation open source de Amazon Dynamo. Les principales fonctionnalités en sont un système complètement distribué, un support de programmation Map/Reduce (langages Javascript et Erlang) permettant de procéder à des traitements distribués à grande échelle. Riak offre un temps de latence faible à des systèmes qui doivent être prêts à monter en charge très rapidement et de façon automatique.

Redis dispose de fonctionnalités plus avancées que Riak et Voldemort en permettant notamment la manipulation des chaînes de caractères ou le stockage de collections. Redis stocke entièrement les données en mémoire vive pour un accès en temps réel très rapide.

Scalaris offre lui assui des fonctionnalités supplémentaires et est similaire à Redis. Il repose sur une implémentation open source de l'Institut Zuse à Berlin et a été un des touts premiers systèmes NoSQL, à respecte les propriétés ACID des transactions distribuées.

3.2.2 Paradigme orienté document

Les systèmes orientés document sont étroitement liés aux systèmes clé-valeur et n'en diffèrent que parce qu'ils permettent une organisation plus complexe des données sous la forme de collections. Ces collections de données appelées documents sont représentées dans des langages descriptifs privilégiant une vision hiérarchique tels que XML ou JSON.[4] Un document est composé de champs et de valeurs associées, ces valeurs peuvent être, soit d'un type simple (entier, chaine de caractère, date, ...), soit elles-mêmes composées de plusieurs couples clé/valeur.. En général, ces systèmes offrent un langage de requête particulièrement évolué qui exploite la structure hiérarchique des documents stockés.[7]

Bien que les documents soient structurés, ces bases de données sont dites *schemaless*. A ce titre il n'est pas nécessaire de définir au préalable les champs utilisés dans un document. La structure interne des documents peut être hétérogène au sein de la base.

Les formats de données JSON et XML servent le plus souvent de langages de sérialisation afin d'assurer l'échange des flux de données entre l'application et la base. les principaux intérêts des systèmes orientés document sont résumés ici :

- leur conception est simple et les bases de données bénéficient de performances élevées.
- les fonctionnalités les plus intéressantes sont liées à la flexibilité du modèle orienté document. Ce modèle est de plus, intuitif pour une grande majorité des usagers et est particulièrement adapté aux applications à visée documentaire, notamment dans le milieu du développement de CMS (*Content Management System*, outils de gestion de contenus).

Les base de données documentaires sont principalement utilisées lorsque une application porte sur différents types d'objets et que les consultations habituelles de la base de données s'effectuent sur différents champs d'un document. La figure 3.2 compare les différentes solutions des systèmes orientés document.

Système	Réplication	Modèle Transactionnel	Passage à l'échelle	Partitionnement automatique	Control concurrence	langage de requêtes
SimpleDB	Async	Théorème CAP(CA)	Horizontal (Read)	Non	None	REST
CouchDB	Multi-Master Async/Sync	BASE	Horizontal (Read)	Oui	MVCC on document	JavaScript Erlang
MongoDB	Master-Slave Async	BASE	Horizontal (Read/Write)	Oui Data Type : BSON	Locks (Write) Atomic operation on fields	JavaScript REST
Terrastor	Sync	Atomicity Consistency Read-Commited	Horizontal (Read/Write)	Oui	Locks	HTTP JSON

FIGURE 3.2 – Comparaison des système orientés document NoSQL

SimpleDB est l'implémentation open source proposée par Amazon. Les avantages de cette solution sont liés à son déploiement léger et sa simplicité d'utilisation. SimpleDB crée automatiquement plusieurs copies géographiquement distribuées de chaque élément de données stocké. Les intérêts en sont une haute disponibilité et une haute durabilité des données.

CouchDB est le projet officiel de la fondation Apache. L'approche met l'accent sur les méta-données, les mécanismes de gestion de versions et l'expressivité de la structure des documents. CouchDB est bien adapté au Web. Il n'offre cependant pas de support en matière de langage de requête non-procédural. Il ajoute donc plus de travail pour le programmeur et nécessite de référencer explicitement les clés.

MongoDB est l'un des systèmes les plus utilisés et en ce sens, a fait l'objet de nombreuses extensions. Il est robuste et mature et propose une gestion avancée de la montée en charge. Ainsi ses capacités de partitionnement et de réplication automatiques permettent de monter en charge horizontalement au fur et à mesure de l'augmentation des besoins. les documents stockés dans une base de données MongoDB sont représentés en BSON (*Binary Serialized document Notation*) qui une sérialisation binaire de JSON.

Terrastore est rendu extensible (développement en Java) au travers d'un système d'événements. Le langage de requête est lui aussi extensible par ce même biais.

Les systèmes orientés documents s'avèrent très similaires mais présentent quelques différences au niveau de leur modèle de données. Ainsi, SimpleDB ne permet pas de stocker de documents imbriqués. De plus, ils utilisent différentes terminologies pour désigner les structures mises en jeu. Par exemple un *Domain* dans SimpleDB est une *base de données* dans CouchDB, et est une *collection* dans MongoDB, ou encore un *Bucket* dans Terrastore.[7]

3.2.3 Paradigme orienté colonne

Les systèmes orientés colonne sont aussi désignés par "systèmes orientés enregistrements extensibles". L'idée est en effet de disposer de structures au format tabulaire mais beaucoup plus flexible que les tables manipulées au sein d'un schéma relationnel et qui vont pouvoir être partitionnées verticalement et horizontalement avant d'être distribuées sur les différents nœuds du système. Un système orienté colonne ainsi est composé d'un ensemble de collections appelées tables. Les tables sont organisées en familles de colonnes qui vont abriter des colonnes qui ne sont pas prédéfinies. L'organisation sous-jacente des données permet à ces systèmes d'être parfaitement dimensionnés pour les traitements massifs d'analyse de données (notamment au travers de mécanismes de type *MapReduce*).
Les systèmes orientés colonne ont les caractéristiques suivantes :

- Ils sont adaptés à des applications qui nécessitent la gestion de différents types d'objets
- Ils offrent des mécanismes qui facilitent la scalabilité horizontale, le partitionnement et la réplication.
- Ils offrent des mécanismes transactionnels qui prennent en charge les accès concurrents et possèdent cet avantage sur les systèmes orienté document.
- Le désavantage porte sur la complexité de l'accès aux valeurs des colonnes si nous comparons avec les systèmes orientés document.

La figure 3.3 compare quelques solutions de ce type de système.

Système	Réplication	Modèle transactionnel	Passage à l'échelle	Partitionnement automatique	Control concurrence	langage de requêtes
HBase	Master-Slave Async	ACID	Horizontal	Horizontal Vertical	Locks/MVCC	REST XML Thrift
HyperTable	Master-Slave Sync	Consistency Durability	Horizontal	Oui	Locks/MVCC	HQL
Cassandra	Master-Master Async	BASE	Horizontal	Oui	MVCC	CQL Thrift
BigTable	Sync+Async	Consistency	Horizontal	Oui	Locks/MVCC	APIcall

FIGURE 3.3 – Comparaison des systèmes orientés colonne

HBase est une implémentation open source de *Google BigTable* et est intégré à l'environnement Hadoop. Chaque enregistrement (row) d'une table peut être accédé directement sur la base de sa clé (opération élémentaire `get`). En outre, des filtres permettent de spécifier des conditions de sélection sur les lignes et les valeurs des colonnes à retourner (par des opérations élémentaires de `scan` sur la totalité de la table ou de `get`).

Les filtres sont exécutés sur le serveur, de sorte qu'ils effectuent réellement des sélections sur la base de données.

Cassandra voit son développement mené sous l'égide de la fondation Apache et est utilisée par Facebook. L'architecture de Cassandra a été conçue pour être complètement distribuée et ne pas présenter de point de défaillance unique (*Single Point Of Failure* ou SPOF). Pour ce faire, tous les nœuds de Cassandra sont équivalents. L'ajout d'un nouveau nœud nécessite alors de ne connaître qu'un seul nœud du *cluster*. Au final, les nœuds sont organisés sous la forme d'un anneau. En cas de défaillance de l'un d'entre eux, il est simplement retiré de l'anneau.

HyperTable est basé sur un modèle développé par Google pour répondre à leurs exigences d'évolutivité et résoudre le problème de passage à l'échelle. Il est bien adapté à un large éventail d'applications. Le partitionnement est plus facile à réaliser avec HyperTable et HBase.

Google BigTable est le système développé par Google App Engine pour un usage interne et n'a pas été rendu disponible à l'extérieur de l'entreprise. Il est à remarquer que ce système a été à la source des autres systèmes à colonne et notamment de HBase et de Cassandra. Google BigTable est considéré comme la preuve de concept (POC ou Proof Of Concept) de ces systèmes.[8]

Les bases de données orientées colonnes diffèrent essentiellement au niveau des mécanismes de concurrence. Par exemple Cassandra fournit un mécanisme de concurrence dit faible (via MVCC [1]) alors que HBase et HyperTable offrent des mécanismes de concurrence plus élaborés (via *Locks* et *Logging*).

3.2.4 Paradigme orienté graphe

Comme nous l'avons évoqué au sein du chapitre précédent, deux catégories de systèmes se rangent parmi les systèmes orienté graphe. Certains considèrent les triplestores RDF comme le moyen le plus performant d'organiser des objets en facilitant la définition de références entre ces objets. L'autre catégorie de système s'articule plutôt sur un modèle de type graphe attribué avec des solutions telles que Neo4J ou OrientDB.
Les deux catégories de système répondent généralement aux critères suivants :

- Traiter des **données fortement connectées**
- Gérer facilement **un modèle complexe et flexible**
- offrir des mécanismes performants pour les lectures locales, par **parcours de graphe**.

Comparativement aux bases de données relationnelles, le cas idéal d'une recherche dans une base orientée graphe est de partir d'un ou de plusieurs nœuds du graphe et de parcourir le graphe. Il est toujours possible d'effectuer des lectures permettant d'accéder à toutes les entités d'un type donné, mais il faut dans ce cas utiliser un système d'indexation. A l'inverse, les bases relationnelles sont tout à fait adaptées à des requêtes `select` sur la totalité d'une table, ainsi qu'à des opérations d'agrégation sur toutes les lignes d'une table. Elles seront cependant moins efficaces sur l'exploitation des jointures entre tables, qui doivent souvent être optimisées par la mise en place d'index portant sur les attributs porteurs des contraintes de clés étrangères. Une base de données graphe offre la possibilité

1. Contrôle de concurrence multi-version

de parcourir les relations entre les nœuds sur la base de pointeurs physiques, alors que les clés étrangères sont des pointeurs logiques.

La modélisation de données sous forme de graphes est naturelle, et comporte donc l'avantage non négligeable d'être lisible, même sans connaissance technique préalable.

Limites

- Les systèmes orienté graphe font état de performances qui se dégradent lorsqu'il s'agit de requêtes globales nécessitant des agrégations mettant en œuvre des calculs sur une large collection de nœuds.

Du point de vue de l'exploitation et la manipulation des données, un système orienté graphe fait appel à : un `langage de définition et de manipulation` des données interne au système, des `interfaces de programmation` et des `interfaces graphiques` à destination des usagers finaux.
Le langage interne au système peut être :[3]

- Un langage de définition de données qui permet de modifier le schéma de la base avec les applications `Add, Change, Delete` ;
- Et/ou un langage de modification de données qui permet de de mettre à jour des données de la base ;
- Et/ou un langage de requêtage pour interroger et restituer les données à l'aide d'une requête.

Par comparaison avec les systèmes relationnels qui fournissent un langage standard et de haut niveau pour la manipulation des données, les systèmes orientés graphe ont souvent recours aux interfaces de programmation. Cela confère plusieurs avantages, notamment un vocabulaire standard pour définir les fonctions et les procédures d'accès et de traitement des données et permettant également un développement unifié des applicatifs associés. En effet, tout est programmation et il n'y a pas d'imbrication d'instructions provenant d'un langage de requête au sein d'un langage hôte. Il n'en demeure pas moins des problèmes qui sont listés ici :

- Une critique principale est la faible capacité d'un système orienté graphe à pouvoir partitionner et répliquer son schéma sur plusieurs serveurs. La scabilité horizontale est en conséquence limitée.
- Il n'y a pas à proprement parler de découplage entre la manière dont les données sont manipulées et la manière dont elles sont organisées (les facilités d'abstraction sont réduites)
- Les langages de programmation peuvent entraîner certaines contraintes et certains biais sur ce qu'il est possible de faire à partir des données
- L'efficacité du système dépend de l'implémentation.

Au-delà de la représentation des données qui différencie les systèmes orienté graphe des autres systèmes NoSQL, ceux-ci se distinguent également par leur capacité à exécuter les algorithmes de graphes courants sur les données qu'ils stockent. Il est ainsi possible de chercher les plus courts chemins d'un nœud à un autre ou encore les nœuds ayant une position centrale dans un graphe.

Le figure 3.4 présente deux systèmes orienté graphe et les compare sur la base de leurs caractéristiques.

Système	Réplication	Modèle transactionnel	Passage à l'échelle	Partitionnement automatique	Control concurrence	langage de requêtes
Neo4j	Master-Slave	Consistency Durability	Horizontal	Non	Locks on write	SparQL Cypher REST
OrientDB	Multi-Master	BASE	Horizontal	Non	MVCC Locks on write	SQL SparQL REST

FIGURE 3.4 – Comparaison des système orientés graphe

Neo4j est un système développé en Java pour être embarqué dans une application multi-plateforme et accessible via une architecture client / serveur et au travers d'une API REST. La manipulation de graphes avec Neo4j est très naturelle à l'aide de son API concise et efficace. *Node* et *Relationship* en sont les principales classes et permettent de modéliser un graphe tout en donnant la possibilité de compléter tout nœud ou relation par un ensemble de propriétés (graphe attribué).

OrientDB est un SGBD document-graphe, permettant le passage à l'échelle, ayant la flexibilité d'une base de données orientée document et la capacité de gérer les liens comme une base de données orientée graphe.

3.3 Triplestores

Les triplestores sont des solutions qui privilégient "une unité format" et qui s'adossent à la mouvance Web de données. Les données sont manipulées sur le Web au format RDF et vont être stockées dans le même format. Comme dans les autres systèmes orientés graphe, il s'agit aussi d'en exploiter la structure en graphe dans la perspective de s'appuyer sur tout ce qui relève de la théorie des graphes. Les modèles RDF sont flexibles et extensibles. Les triplestores proposent des fonctionnalités sur la base du méta-modèle RDF qui vont faciliter le stockage des graphes et des triplets qui y sont contenus. En général, des structures d'index de type arbre équilibré (BTree+) sont définies sur la base des patrons de triplets les plus à même d'être exploités dans les requêtes.[25]

3.4 Discussion sur les différents types SGBD

Nous terminons ce chapitre par une discussion générale qui reprend les avantages et les inconvénients de chacune des grandes orientations en terme de stockage de données. Certains avantages vont se révéler déterminants dans le choix de retenir tel ou tel système pour la gestion de gros volumes de données spatio-temporelles. Nous séparons la discussion en prenant en considération différents aspects qui à notre avis reflètent les principales qualités que doit recouvrir tout système de gestion de données.

- **Modèle de données standard et flexible** : Les systèmes NoSQL ont pour certain d'entre eux un modèle de données spécifique qui oblige tout modélisateur à se familiariser avec ces différentes manières d'organiser l'information. Les systèmes orientés clé-valeur, document et colonne vont donc nécessiter un temps de prise

en main. Une première difficulté est de faire migrer les données d'un système à un autre. Une deuxième difficulté est d'intégrer des données provenant de différents systèmes, par exemple provenant d'un système orienté document et d'un système orienté colonne. L'approche qui consiste à créer des systèmes volontairement novateurs et qui s'affranchissent des standards, a l'intérêt d'ouvrir le champ des possibles mais va se révéler pénalisant si nous nous plaçons dans une vision globale de partage et d'échange de données qui ne relève pas d'un contexte particulier.
Dans ce sens, les systèmes de gestion de triplets RDF se posent comme des candidats intéressants avec des formats de description de données RDF / RDFS et OWL normalisés par le W3C. Le langage RDF est exploité comme un métamodèle dans les activités liées à la définition de sources de données liées et ouvertes sur le Web (Linked Open Data) et de nombreuses communautés dans les sciences du vivant et de la santé ont adopté ces nouvelles manières de partager l'information. RDF peut décrire toute ressource au travers de ses propriétés intrinsèques ou de ses relations avec d'autres ressources. Ces descriptions vont pouvoir évoluer dans le temps et être explicitées de manière décentralisée par rapport à la ressource elle même. Ainsi une application va pouvoir consulter plusieurs triplestores de manière à satisfaire les besoins de requêtes complexes nécessitant l'intégration de différentes données. Les systèmes orientés document sont en directe filiation avec les systèmes de gestion de données semi-structurés (SGBD XML), et sont pensés de manière à organiser des documents dotés de structures différentes. L'idée là aussi est de gérer des structures arborescentes qui peuvent être complexes, évolutives et diverses. Les documents sont par ailleurs définis et manipulés dans des langages qui rappellent XML. Les systèmes orientés colonne misent aussi sur l'évolutivité et l'extensibilité des tables et enregistrements qui constituent leurs schémas. Tous ces systèmes se démarquent des systèmes relationnels par leur capacité à laisser les schémas de données ouverts. Un reproche qui peut cependant leur être fait est de bien souvent privilégier un point de vue en particulier dans ces modèles ouverts et de construire les applications autour de ce point de vue. Dans notre contexte, nous pourrions avoir une entrée dans l'information par exemple par les entités géographiques.

– **Un langage de requête standard et expressif** : Les SGBD NoSQL ne fournissent pas toujours une langage de requête de haut niveau, qui pourrait être vu comme un équivalent du langage SQL. L'interrogation des bases de données se fait alors en s'appuyant sur un modèle de données spécifique, soit avec un langage de requête simplifié, soit au travers d'un langage de programmation. Dans le cas de SimpleDB ou GQL, un langage de requête est nouvellement défini pour interroger et manipuler les données stockées dans ces systèmes en particulier.
Le langage de requête SPARQL est à l'inverse devenu un standard du W3C et est soumis régulièrement à de nouvelles évolutions de manière à prendre en charge un grand nombre de fonctionnalités. Il offre ainsi une grande richesse d'expression dans la manière d'interroger les graphes RDF et exploite pour ce faire la définition de patterns sur les triplets. Il intégre également des fonctions de calcul (agrégation, comptage, moyenne, ...). SPARQL est donc devenu le langage de requête exploité par l'ensemble des triplestores. Dans notre contexte d'étude, il est à remarquer que SPARQL fait l'objet d'une extension vers le spatial avec la

définition de GeoSPARQL.

– **Formats de sérialisation des données normalisés** : Les fonctionnalités rendues disponibles pour importer et exporter les données d'un système de gestion de données sont également un critère de choix important dans un contexte de forte volumétrie de données. Les SGBDR ont des fonctionnalités de chargement qui sont appelées des *fonctions de dump de tables* (SQL Loader pour Oracle) et qui facilitent le chargement de gros volumes de données au format CSV (Comma Separated Value). Ils possèdent également des fonctionnalités qui permettent d'exporter le schéma et/ou les données d'une instance de base de données (fonctions PL/SQL du paquetage DBMS_Metadata pour Oracle). Les systèmes NoSQL disposent également de fonctionnalités d'import/export de données et les données sont échangées au travers des formats classiquement exploités dans le contexte du Web et des bases de données tels que XML, JSON et CSV. Ainsi HBase donne accès à une implémentation Map/Reduce pour l'import de fichiers de données au format tabulé. Les triplestores peuvent de leur côté tirer parti pour l'import/export de formats de sérialisation adaptés tels que N-Triples et N-Quads.

Les éléments importants à même d'orienter le choix d'un système de persistance de données sont fixés et il convient maintenant d'opérer ce choix et de définir au préalable les modèles conceptuels de données, les mieux à même de prendre en charge les dimensions spatiales et temporelles des objets d'intérêt. Nous nous sommes dotés d'un cadre de travail qui comporte un jeu de données spatio-temporelles d'étude, d'une reprise de l'existant en terme de modèles de données pour le spatial et le temporel et de tests préliminaires d'implémentation. Le cœur de notre proposition porte sur la définition de modèles de connaissances spatio-temporels qui vont faciliter la gestion et l'interprétation de données spatio-temporelles, et en particulier de trajectoires d'objets mobiles (bâteaux ou avions) et de comportements (en particulier déplacements) qui pourraient apparaître comme inhabituels.

CHAPITRE 4

APPROCHE MÉTHODOLOGIQUE

L'essor des technologies autour de la communication sans fil, l'évolution des systèmes de référencement spatial comme temporel, et des systèmes d'information géographiques (SIG) ont favorisé l'apparition de nouveaux types de services et d'applications liés à la mobilité et à la localisation d'entités d'intérêt. Les divers usages de ces nouvelles activités apparaissent dans des secteurs variés et vont de la gestion de flottes de véhicules, à du transport urbain ou maritime ou bien à de nouveaux services liés à la mobilité : informations touristiques, cartes routières, implantations commerciales, ...

La modélisation, la gestion et le traitement de gros flux de données spatio-temporelles offrent des perspectives de recherche d'intérêt et mobilisent notamment les communautés de chercheurs travaillant autour des systèmes d'information et des systèmes à base de connaissances. Les problèmes de recherche ouverts portent notamment sur la complexité des données à modéliser qui sont, de plus, très souvent mobiles et changeantes. Ces données sont de plus multiples et hétérogènes et sont acquises au travers de nombreux capteurs bien souvent équipés de systèmes de positionnement global de type GPS et qui sont distribués sur la surface terrestre (équipements embarqués et nomades). Par exemple, les systèmes d'information actuels et notamment les systèmes d'information géographiques ne proposent pas de fonctionnalités spécifiques concernant la gestion d'objets mobiles.

A cet effet, notre approche se consacre à la définition de modèles de données et de connaissances appropriés à la prise en considération des dimensions temporelles et spatiales que peuvent revêtir des objets d'intérêt, quelle que soit la thématique considérée. Nous travaillons sur un cas d'étude concret qui porte sur des scénarios de transports maritimes et de déplacements de bateaux. Nous reprenons différents travaux de recherche autour de la modélisation et de l'échange de données spatiales comme temporelles et nous les étendons pour répondre à la spécificité de la représentation d'objets mobiles et de leurs trajectoires au travers du temps. D'une part, le modèle ontologique spatial GeoSPARQL et d'autre part le modèle ontologique temporel OWL-Time (défini dans le cadre des activités autour des web services sémantiques OWL-S) sont ainsi mis à profit. Le modèle de données spatio-temporel dégagé est restitué sous forme de diagrammes de classes et de diagrammes d'objets UML. Les modèles conceptuels sont également partiellement déclinés en langage OWL. L'évaluation et la validation du travail de modélisation sont permises par la mise en place d'un système de gestion de triplets RDF qui vient faciliter la simulation des comportements, dans le temps et dans l'espace d'objets mobiles en très grand nombre.

4.1 Jeux de données d'étude

Dans le domaine de la supervision maritime, le jeu de données réel est jugé sensible et en conséquence, n'est pas mis à notre disposition.
Nous ne disposons donc que d'informations partielles qui portent, par exemple, sur la volumétrie des bateaux et sur leurs trajectoires. Ainsi, nous savons que les activités de supervision traitent environ 100 000 entités (en général des bateaux) et qu'un demi milliard de données sont mises à jour quotidiennement. Chaque entité est décrite par quelques dizaines d'attributs et pour l'instant, seule la position de l'entité et la date du relevé sont conservées. Les relevés s'effectuent toutes les heures. Nous listons ci-dessous des exemples de règles d'alertes pour les comportements qui sont jugés anormaux :

- présence des navires hors des routes maritimes.
- navires dont les positions sont incohérentes (ex : sur la terre ferme), ou dont le rapport route/vitesse, calculé par rapport à la dernière position, semble irréaliste.
- navires dont la vitesse est inférieure à XX nœuds et dont le plat-bord est inférieur à XX mètres et pressente dans une zone de piraterie.
- navires civils ayant récemment changé de nom, de pavillon, ou de port d'attache.
- navires en période d'entretien de longue durée
- navires civils d'intérêt militaire

De manière à nous rapprocher le plus possible du jeu de données de superversion de trajectoires maritimes, nous avons construit un jeu de données spatio-temporelle acquis à partir d'un logiciel installé sur smartphone qui enregistre les localisations du smartphone (et donc de la personne qui le possède) à l'aide d'un référentiel de positionnement de type GPS. Plusieurs personnes ont ainsi enregistré leurs allées et venues quotidiennes et ce, sur plusieurs jours, sous format `csv`. Chaque fichier `csv` est considéré comme un objet mobile. Le figure 4.1 illustre une portion des données enregistrées.

FIGURE 4.1 – Une portion de notre jeu de données spatio-temporelles

4.2 Adossement à des travaux existants

De nombreux travaux sont menés depuis quarante ans, avec pour objectif principal, la mutualisation d'une multitude de jeux de données géographiques. Les enjeux sont divers et variés, et la maîtrise de l'information géographique est une des priorités des gouvernements notamment en qui concerne la gestion des territoires. L'étude des phénomènes spatio-temporels nécessite d'approfondir les notions relatives aux deux dimensions : espace et temps. Dans ce qui suit, nous commençons par donner un aperçu général sur les travaux portant sur chacune de ces dimensions et nous en expliquons les principes clés. Nous ne décrivons ici que certains des travaux conduits au sein du consortium de l'Open GIS ou OGC, sur lesquels nous appuierons par la suite. Nous nous intéressons, dans un second temps, à l'ontologie OWL-Time pour ce qui concerne le temps.

4.2.1 OGC ou Open GeoSpatial Consortium

L'OGC[20] est une organisation internationale, qui réunit depuis 1994, de nombreux acteurs (compagnies privées, universités et autres organismes de recherche, agences gouvernementales), de manière à définir communautairement les standards et interfaces utiles à la mise en œuvre de solutions web interopérables autour du géospatial. Ainsi une spécification abstraite et une architecture de services web sont mis à la disposition du grand public par l'OGC[22]. Nous abordons exclusivement les standards de description de données qui vont permettre d'accéder aux données, de les partager et de les réutiliser à large échelle et ainsi répondre aux problèmes d'interopérabilité des systèmes d'information géographique (SIG) tout en les complétant.

GML (Geography Markup Language)[21] est un des standards de description des données de l'OGC. Il est défini en XML schéma et s'appuie sur une série de standards, à l'exemple de ISO 19115 qui couvre le champ des métadonnées pour le géospatial ou de ISO 19111 qui structure tout ce qui touche au référencement spatial et aux systèmes de référencement par coordonnées (CRS ou *Coordinate Reference System*). De même, GML reprend à son compte certaines entités de la spécification abstraite de l'OGC à l'exemple des entités spatiales d'intérêt (ou *geographic features*), de leurs géométries ainsi que des relations topologiques qui se nouent entre elles. L'objectif est de disposer d'une organisation de l'information géographique unifiée et de faciliter les échanges de données entre applications.
Nous présentons ici quelques notions nécessaires à la compréhension des modèles.

4.2.2 Notion de feature

La notion de *feature* que nous pourrions traduire par entité caractéristique est une notion centrale pour l'OGC et est définie au sein du standard ISO 19101 comme une abstraction d'un phénomène présent dans le monde réel [1]. Une *geographic feature* est un *feature* plus spécifique et une entité caractéristique qui est associée à une position relative sur la Terre. L'état d'une *feature* est défini par un ensemble de propriétés vues sous forme de triplets {nom, type, valeur}. Les *features* qui sont donc des objets ou des évènements,

1. A feature is an abstraction of real world phenomena (ISO 19101)

localisés sur la surface terrestre vont, par exemple, pouvoir être décrites au travers de propriétés géométriques et être également enrichies par des métadonnées.

4.2.3 Système de référencement par coordonnées

Le positionnement géographique d'une entité spatiale se doit d'être rigoureux et nécessite à cet effet l'emploi d'un système de référencement spatial. Différents référentiels existent et se distinguent par la manière de représenter des objets dans l'espace et notamment de les situer à l'aide de couples de coordonnées géographiques.

L'OGC définit un système de référencement par coordonnées comme un système de coordonnées muni d'au moins deux dimensions, relié à la surface terrestre via une de ses représentations appelées *datum* (ou ellipsoïde qui détermine la taille et la forme de la Terre). Le système de coordonnées de référence fournit un contexte pour les géométries des objets d'intérêt afin d'en définir les positions et d'établir des relations entre des objets vus comme des collections de coordonnées.[6] La combinaison d'un système de coordonnées, d'un ellipsoïde ou datum, et d'un système de projection définissent un CRS. Le répertoire EPSG (European Petroleum Survey Group) [2] compile les systèmes de coordonnées les plus utilisés.

L'OGC facilite, par ailleurs, l'identification des systèmes de coordonnées en les organisant sous forme de ressources étiquetées au sein d'un vocabulaire au format rdf (espace de nommage : http ://www.opengis.net/def/crs/). Deux institutions font autorité : EPSG et OGC. Ainsi le système de référencement global WGS84 (préférentiellement exploité au sein des GPS) a pour URI

```
http://www.opengis.net/def/crs/OGC/1.3/CRS84
```

De même le système de référencement Européen (European Terrestrial Reference System 1989) si l'autorité est l'EPSG, est identifié par :

```
http://www.opengis.net/def/crs/EPSG/0/4937
```

4.2.4 Données spatio-temporelles

Une entité spatio-temporelle est une représentation d'une entité du monde réel, d'une part composée d'une identité qui décrit une sémantique fixe de l'entité, et d'autre part composée de propriétés descriptives et spatiales qui peuvent varier dans le temps et qui constituent la partie dynamique de l'entité [14]. Si l'identité même d'une entité est soumise à une variation, alors l'entité spatio-temporelle est considérée comme une nouvelle entité. Nous voyons donc qu'une entité spatio-temporelle est de nature hautement évolutive.

Deux principaux types d'entités spatio-temporelles sont retrouvées dans la littérature :[15]
1) Les **objets mouvants**, comme par exemple un navire sillonnant les mers,
2) Les **objets changeants**, par exemple, un port maritime dont le périmètre évolue au cours du temps.

2. http ://www.epsg-registry.org/

4.2.5 Langage GeoSPARQL de l'OGC

GeoSPARQL est une extension spatiale des langages RDF et SPARQL pour faciliter les usages (en particulier définition et consultation) des données géographiques, dans le contexte des sources de données liées et du Web sémantique. GeoSPARQL propose en ce sens des représentations adaptées aux entités spatiales, à leurs propriétés et aux fonctions de calcul (par exemple topologiques ou métriques) qui s'y appliquent. Le langage GeoSPARQL est construit de manière modulaire et comprend différents composants parmi lesquels :

- ***Core*** qui est un modèle de connaissances au format OWL, décrivant les entités spatiales de haut niveau, et qui s'inspire de modèles de données existants, en particulier de *Simple Features Model* [19]. La superclasse dans Core est `geo:SpatialObject`. *geo* étant le préfixe de l'ontologie Core par convention de nommage.
  `geo:SpatialObject` est spécialisée par `geo:Feature` et `geo:Geometry`.
  `geo:Feature` correspond à l'entité Feature définie précédemment au sein du standard ISO 19101 et va jouer le rôle de superclasse pour toutes les classes thématiques `Feature` que les usagers seront amenés à construire. L'idée est véritablement de fournir une ontologie cadre qui va pouvoir être étendue en fonction des besoins fonctionnels de chaque communauté faisant appel à de l'information géographique.
- ***Extension du vocabulaire topologique*** :
  l'association réflexive `geo:hasSpatialRelation` qui permet de relier deux ou plusieurs `geo:SpatialObject` est raffinée dans l'ontologie. Une hiérarchie de propriétés (`owl:objectProperty`) permet ainsi de prendre en charge différentes familles de relations topologiques à l'exemple de RCC8 ou Simple Features.
- ***Extension géométrique*** :
  la classe `geo:Geometry` est étendue pour prendre en charge la diversité des objets géométriques, à l'exemple de `geo:point`, `geo:line`, `geo:polygon`,..
- ***Topologies géométrique*** :
  Un ensemble de fonctions SPARQL étendues à la prise en charge de calculs spatiaux, concernant en particulier, les relations non topologiques.
- ***Ensemble des règles au format RIF***[3] pour les transformations des requêtes.

GeoSPARQL propose donc un ensemble de modèles et de fonctionnalités particulièrement étoffé. De plus, le principe de modularité permet d'étendre les composants proposés en fonction des besoins applicatifs des communautés visées. Nous nous concentrons dans un premier temps sur le composant Core et ses trois classes de haut niveau.

- `geo:SpatialObject`
- `geo:Feature`
- `geo:Geometry`

Modèle Core

Nous proposons, à la figure 4.2, un diagramme de classes UML qui reprend les principaux éléments du modèle Core. La classe `geo:Feature` fait office de point d'exten-

3. Rule Interchange Format

sion. Nous exploiterons cette caractéristique pour étendre `geo:Feature` avec nos classes d'intérêt.

Les classes `geo:Feature` et `geo:Geometry` sont disjointes et héritent de `geo:SpatialObject`. Le choix est fait de distinguer une entité de ses géométries, de manière à donner de la flexibilité dans le calcul, par exemple des relations topologiques ou métriques entre objets géométriques. De plus, l'entité va pouvoir être caractérisée d'un point de vue thématique. Ainsi, une *feature* est tout simplement une entité du monde réel, qui pourrait être un parc, un aéroport, un monument ou encore un restaurant. Une *geometry* représente toute forme géométrique, par exemple un point, un polygone, ou une ligne, et va être une représentation de la position spatiale d'une *feature* dans un référentiel donné (*CoordinateReferenceSystem*).

A cet effet, `geo:Feature` et `geo:Geometry` sont associés au travers de la relation `geo:hasGeometry`. En terme de multiplicité, un objet de type `geo:Feature` peut être associé à de 0 à n objets `geo:Geometry`. Un objet `geo:Geometry` est à l'inverse associé à un seul objet de type `geo:Feature`. Une relation `geo:hasDefaultGeometry` vient spécialiser `geo:hasGeometry` et permet d'attribuer une géométrie par défaut à une entité spatiale. Un objet de type `geo:Geometry` est associé à seul référentiel géographique et donc à un et un seul objet `CoordinateReferenceSystem`.

Attributs et caractéristiques de la classe `geo:Geometry`

La classe `geo:Geometry` possèdent différents attributs qui s'apparentent à la métadonnée venant qualifier une géométrie. Ces propriétés sont énumérées ci-dessous :

1. `geo:dimension` correspond à la dimension topologique de l'objet géométrique et doit être inférieure ou égale à la dimension de coordonnées. Dans les collections non homogènes, il retourne la plus grande dimension topologique des objets contenus.
2. `geo:coordinateDimension` définit les coordonnées d'un objet géométrique par rapport aux axes X et Y ou Z. Il peut s'agir de coordonnées 2D ou 3D.
3. `geo:spatialDimension` correspond à la dimension de la partie spatiale de la propriété `geo:coordinateDimension` utilisée dans la définition de l'objet géométrique.
4. `geo:hasSerialization` associe une géométrie définie au format RDF à une de ses sérialisations textuelles possibles. Cette sérialisation est basée soit sur le standard WKT [4], ou bien sur le standard GML [5].

RDFS Datatypes et sérialisation

Partager et échanger des objets géométriques est un point important. Pour cela, des formats simples et textuels sont rendus disponibles dans GeoSPARQL. Ainsi deux types de données RDFS nommés `geo:wktLiteral` et `geo:gmlLiteral` permettent de disposer d'une représentation littérale de type chaîne de caractères conforme respectivement aux standards WKT ou GML, pour la manipulation des objets géométriques. Deux propriétés vont faciliter la prise en charge du littéral dans un format ou dans l'autre.

4. Well Known Text (déjà exploité au sein de Simple Features (ISO 19125)
5. Geography Markup Langage

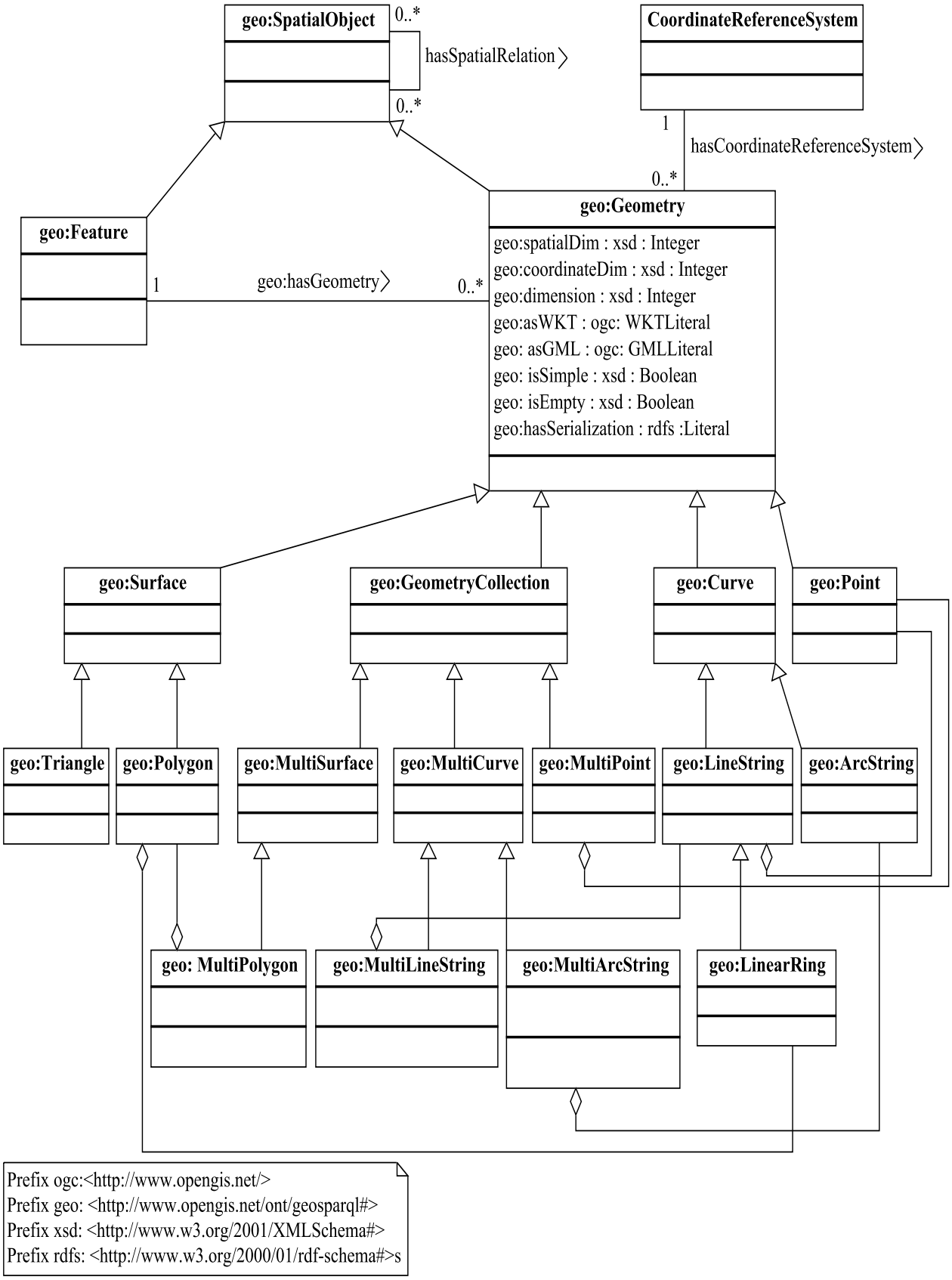

FIGURE 4.2 – Diagramme de classes UML illustrant le modèle Core de GeoSPARQL

1. `geo:asWKT` : La propriété `geo:asWKT` associe un objet géométrique à une valeur littérale de type `geo:wktLiteral` qui comprend les informations de géométrie réelle. Par exemple, la figure 4.3 restitue la valeur littérale d'un object géométrique de type point. La latitude et la longitude sont données et associées à leur référentiel : le système de coordonnées de référence géodésique longitude-latitude WGS84 qui est le système par défaut dans Simple Features 1.0.

```
"<srsName=\"http://www.opengis.net/def/crs/OGC/1.3/CRS84\>
 Point(33.95 -83.38)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
```

FIGURE 4.3 – Exemple de sérialisation WKT d'un objet de forme point

2. `geo:asGML` : La propriété `geo:asGML` associe un objet géométrique à une valeur littérale de type `geo:gmlLiteral`. Un élément géométrique conforme au schéma GML est attendu (ici `gml:Point`). La figure 4.4 illustre l'encodage du même objet géométrique décrit dans la figure 4.3, cette fois-ci dans le format GML.

```
"<gml:Point
     srsName=\"http://www.opengis.net/def/crs/OGC/1.3/CRS84\"
     xmlns:gml=\"http://www.opengis.net/ont/gml\">
   <gml:pos>-83.38 33.95</gml:pos>
 </gml:Point>"^^<http://www.opengis.net/ont/geosparql#gmlLiteral>
```

FIGURE 4.4 – Exemple de sérialisation GML d'un objet de forme point

Relations spatiales

Nous avons précédemment fait mention de l'association réflexive `hasSpatialRelation` qui permet de relier au moins deux `geo:SpatialObject`. Cette association revêt une importance particulière car elle va permettre de prendre en charge tout type de relation spatiale que cela soit entre des objets de type `geo:feature` ou de type `geo:geometry`. Elle va par la suite, faciliter les comparaisons et les calculs entre ces objets. De manière synthétique, nous pouvons classer les relations spatiales qui viennent spécialiser `hasSpatialRelation` selon trois catégories[9] : les relations topologiques [6], les relations de calcul et les relations métriques.

Nous proposons un diagramme de classes 4.5 qui illustre l'arborescence des relations. A cet effet, les associations de type `owl:objectProperty` ont été choisies. `geo:SpatialRelation` est la classe de plus haut niveau. Nous y avons rajouté des classes intermédiaires (préfixe my) de manière à mieux distinguer les différentes relations. Trois manières de percevoir les relations topologiques, donnant lieu à trois vocabulaires différents, sont pris en charge (voir quelques unes des sous-classes de GeometryTopologyExtension au sein de la figure 4.5). Ces relations vont admettre des valeurs booléennes pour résultats et offrent une représentation qualitative de la topologie :

6. La topologie est l'ensemble des relations qui nous permet de percevoir et situer les objets les uns par rapport aux autres.

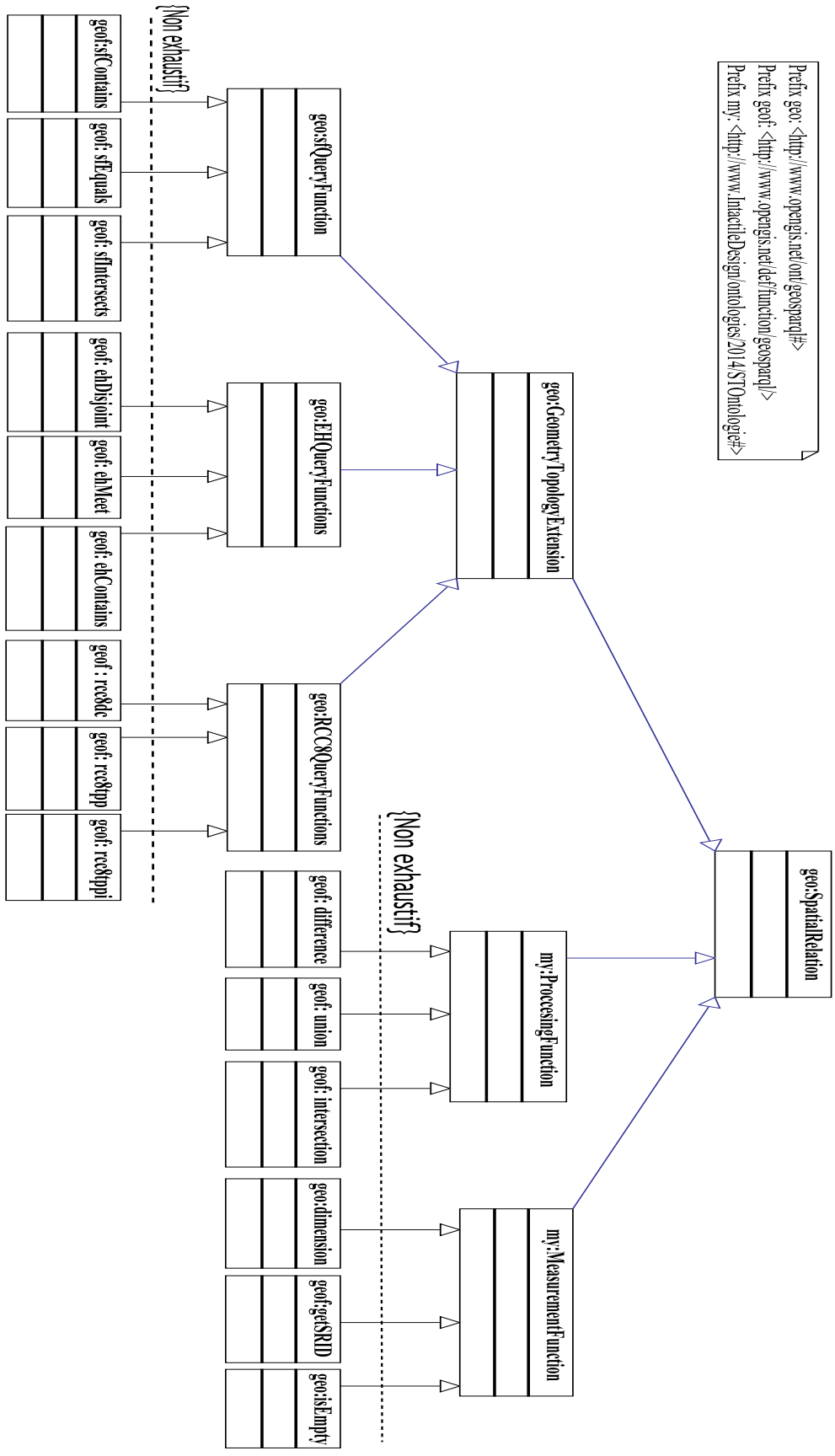

FIGURE 4.5 – Hiérarchie des relations spatiales

– **Modèle Simple Feature de l'OGC**[17]
Simple Features Access et *OGC Filter Encoding Implementation standards* fournissent des noms typiques pour les opérateurs topologiques, à l'exemple de `geo:sfContains` qui permet d'expliciter une notion de contenant/contenu entre deux objets.

– **Modèle d'Egenhofer dit à 9-intersection** [7], standard s'attachant à décrire les relations spatiales entre deux objets sous la forme d'une matrice à deux dimensions. Chaque objet est considéré sous l'angle de son intérieur, de son extérieur et de sa frontière. Deux objets comparés vont être évalués au travers de ces trois caractéristiques. Les valeurs possibles sont T pour vrai, F pour faux, et * pour pas d'explication(). Les cellules d'une matrice DE-9IM explicitent donc les intersections possibles entre les objets. Dans l'exemple proposé en Figure 4.6, la relation topologique `geo:ehContains` est définie au travers de la matrice avec pour signature (pattern) TTTFFTFFT.

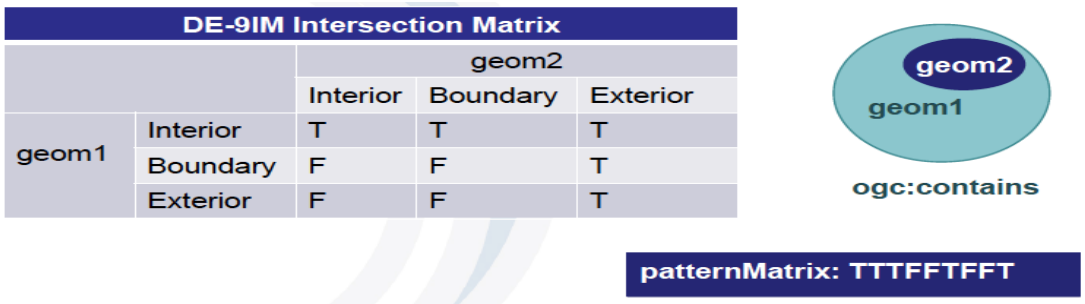

FIGURE 4.6 – Illustration autour de la relation topologique `geo:ehContains` de Egenhofer et matrice DE-9IM associée

– **Modèle RCC8** [8], bâti sur huit relations méréotoplogiques qui viennent modéliser les positions relatives des objets les uns par rapport aux autres. Chaque objet x est défini comme possédant une frontière notée δx et un intérieur noté $\mathring{x}$. L'évaluation qualitative va porter sur l'intersection et la différence au sens ensembliste des parties communes (intérieur et/ou frontière). Par exemples deux objets notés x et y vont satisfaire la relation d'équivalence notée `geo:rcc8eq` si il est montré que $\mathring{x}$ - $\mathring{y} = \emptyset$ et que $\mathring{y}$ - $\mathring{x} = \emptyset$. La figure 4.7 offre une illustration exhaustive de RCC8.

D'autres types de relations prennent plusieurs objets géométriques en entrée et produisent soit une nouvelle géométrie en sortie (ProcessingFonction au sein de la figure 4.5), ou bien un autre type de résultat quantitatif (MeasurementFunction au figure 4.5). De nombreux travaux portant sur les relations spatiales, sont disponibles dans la littérature et nous ne reprenons pas ici tout ce qui concerne les relations métriques (de distance et de direction).

7. Dimensionally Extended nine-Intersection Model (DE-9IM)
8. Region Connection Calculus

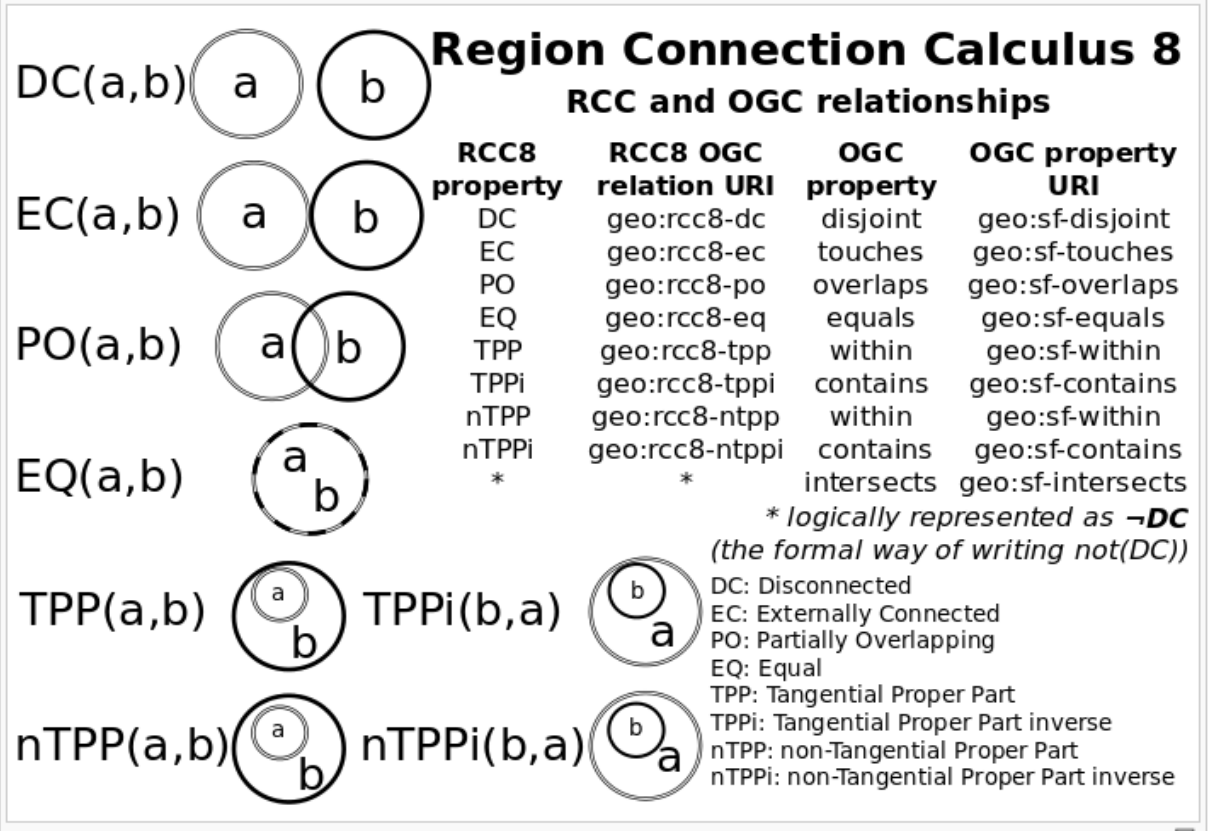

FIGURE 4.7 – Synthèse des relations RCC8 et convention de nommage OGC
`extrait de en.wikipedia.org/wiki/Region_connection_calculus`

4.2.6 Notre proposition d'extension du modèle Core de GeoSPARQL

GeoSPARQL propose donc un modèle qui offre la possibilité de représenter de manière simple des entités, leur positionnement géographique et leurs géométries au travers des classes `geo:Feature` et `geo:Geometry`. Nous étendons la classe `geo:Feature` pour prendre en charge nos entités d'intérêt dans le contexte de la navigation maritime et de façons plus large d'objets mobiles. L'extension est proposée dans la figure 4.8 sous la forme d'un diagramme de classes et nous avons également défini un espace de noms propres, qualifié par `myS` pour bien différencier notre proposition, de l'existant. Les ajouts sont présentés en couleur verte dans la figure.

Un objet mobile tel qu'un navire, un avion ou encore un oiseau est représenté au travers d'une classe nommée `myS:MovingObject` et possède différent attributs comme un identifiant `myS:objectId`, un nom `myS:objectName`, un type `myS:type` et une description `myS:description` qui explique sa situation actuelle. Tout objet mouvant est de plus lié à au moins une géométrie grâce au lien `geo:hasGeometry` qui décrit sa forme et ses attributs spatiaux. Les navires sont considérés comme ayant une géométrie de type `geo:Point`. Pour pouvoir rajouter des attributs à cette géométrie, nous avons créé une nouvelle classe venant spécialiser `geo:Point` qui a pour nom `myS:MovingPoint`. Les attributs suivants sont décrits : un identifiant `myS:pointId`, une altitude `myS:pointAltitude`, une direction

myS:pointDirection et une vitesse myS:pointSpeed qui sont les données existantes dans notre jeu de données. Donc un objet mobile est considéré comme défini par plusieurs géométries qui sont des objets myS:MovingPoint ordonnés au cours du temps.

L'entité nommée myS:Way définit les chemins maritimes précisés par les services de surveillance et est liée à une géométrie geo:LineString. L'entité myS:Port définit les aires de stationnement des navires et est liée à une géométrie geo:Polygon. Ces deux entités pourraient aussi venir spécialiser la classe myS:MovingObject mais nous nous concentrons sur les entités myS:MovingObject et myS:MovingPoint.

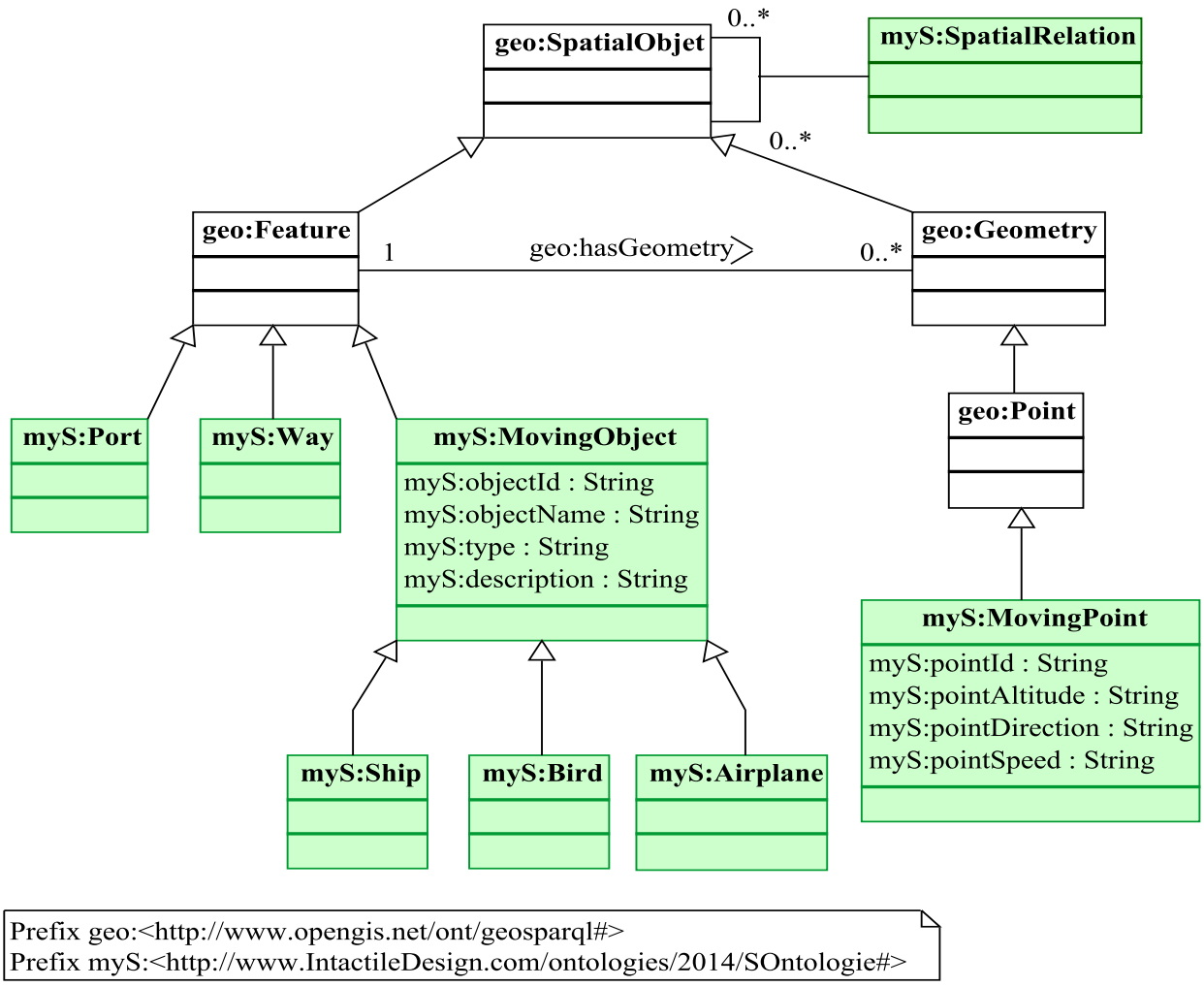

FIGURE 4.8 – Notre proposition d'extension de l'ontologie GeoSPARQL

Un diagramme d'objets en figure 4.9 est également proposé et vient illustrer le diagramme de classes à partir de données provenant de notre jeu de données.

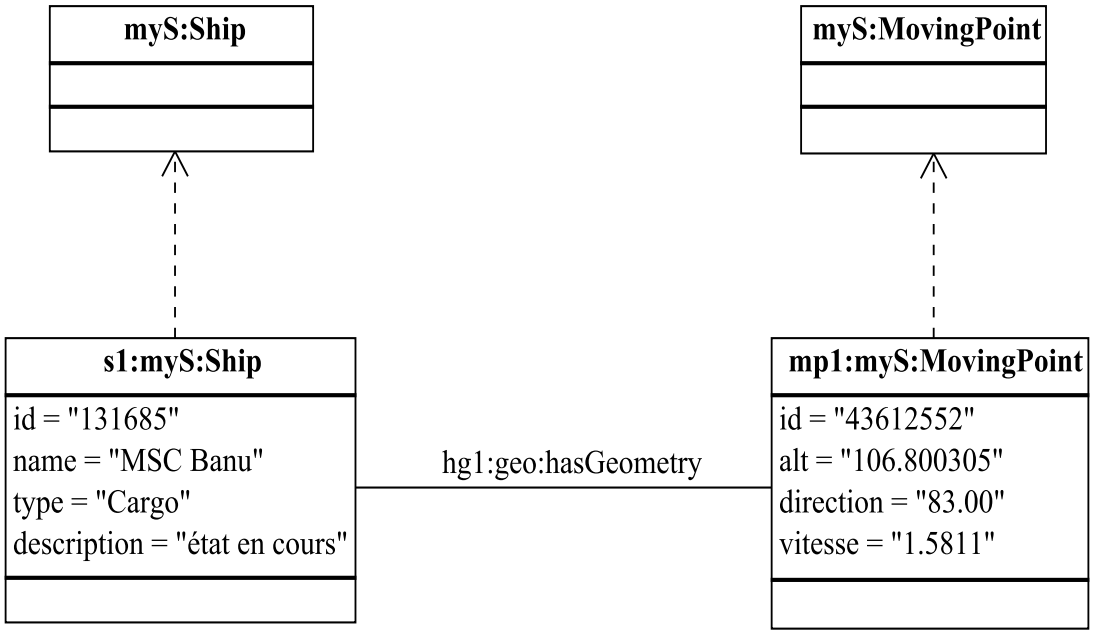

FIGURE 4.9 – Diagramme d'objets illustré au figure 4.1 à partir de modèle étendu proposé en figure 4.8

4.2.7 Une ontologie OWL pour représenter le temps

L'espace, les relations dans l'espace et les changements dans l'espace sont des notions fondamentales en géographie. La notion de changement sous-entend une dimension temporelle.

Le projet DAML [9] a proposé une ontologie du temps (OWL-Time décrite conceptuellement au sein de la figure 4.10) afin de modéliser la dimension temporelle présente en filigrane dans le contenu des pages Web et dans le déroulement chronologique de l'exécution de services Web. Cette ontologie organise les propriétés topologiques des instants, des intervalles, des mesures de durée, et explicite les notions d'horloge et de calendrier pour la représentation des dates.

Les concepts temporels sont classés en plusieurs catégories et chaque catégorie va être décrite au travers de ses principaux prédicats et propriétés.[18]

Relations topologiques temporelles

1. **Instants et Intervalles**
   La classe *TemporalEntity* possède deux sous-classes qui sont *Instant* et *Interval*.
   Les intervalles sont des objets mesurés et les instants sont représentés comme des points élémentaires et en ce sens, ne peuvent pas être décomposés en d'autres points.
   Pour plus de facilité, nous pouvons dire que le début et la fin d'un instant sont

9. Le projet DARPA ou Agent Markup Language, est une initiative associée au Web sémantique, à partir de laquelle les agents humains et automatiques peuvent accéder aux données et traitements sur le Web au travers de descriptions du contenu et des fonctionnalités des ressources visées plutôt qu'au travers de simples mots clés[18]

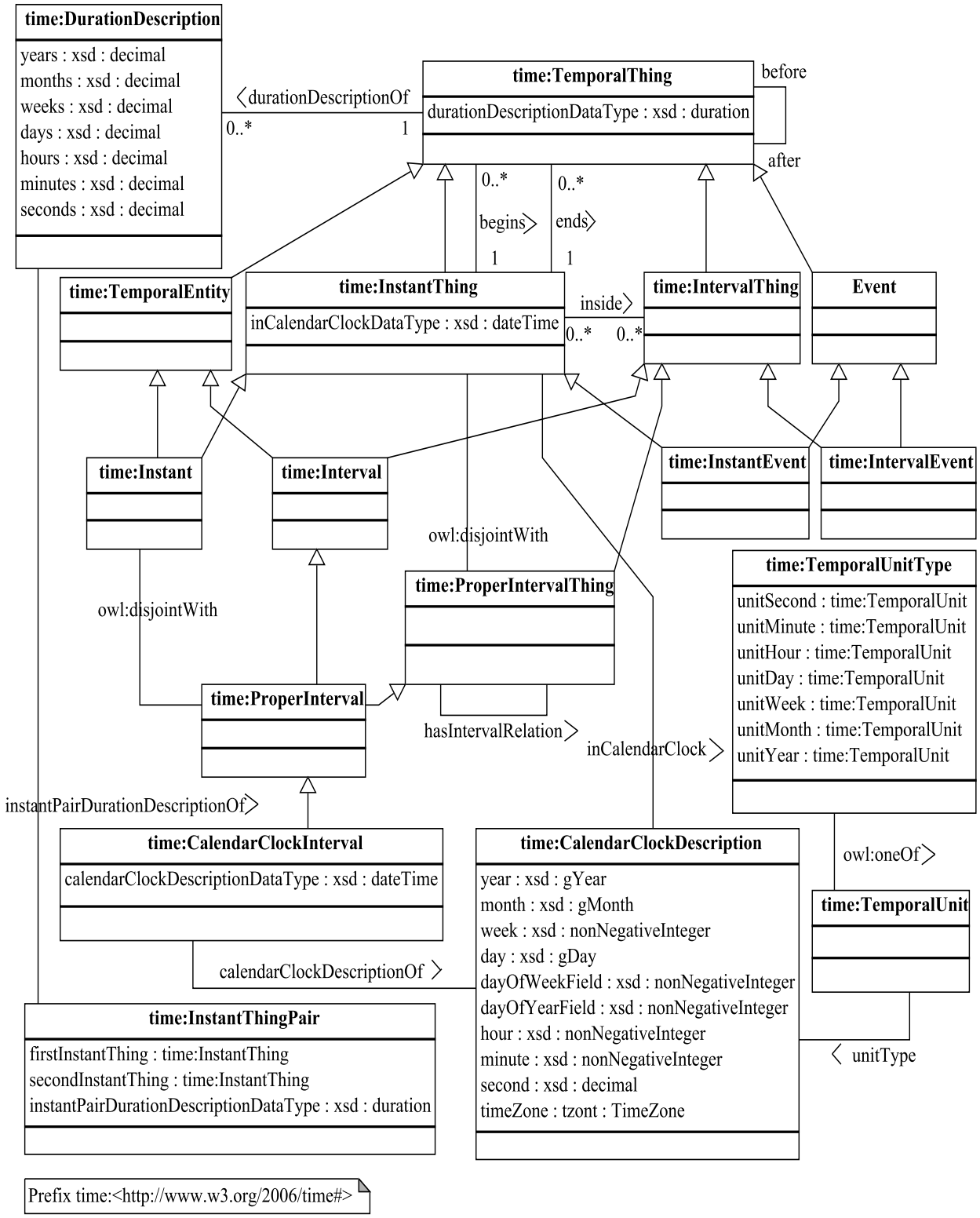

FIGURE 4.10 – Modèle conceptuel illustrant l'ontologie du temps définie dans le cadre du projet DAML

identiques à l'instant lui même .
Les prédicats *begins* et *ends* sont des associations qui lient *Instant* à *TemporalEntity*. Pour ce qui concerne les intervalles infinis, un intervalle infini positif n'a pas de fin et un intervalle infini négatif n'a pas de début. Par conséquent, dans l'ontologie de base, nous utilisons les relations *begins* et *ends* plutôt que de redéfinir de nouvelles fonctions *beginingOf* et *endOf*, car l'information ne serait pas toujours disponible. Elles peuvent être définies dans une extension de l'ontologie de base qui utiliserait les instants infinis positifs et négatifs. Le prédicat *inside* est une relation entre un instant et un intervalle et n'est destiné qu'à inclure des débuts et des fins à des intervalles. Ce concept sera utile pour caractériser les notions d'horloge et de calendrier et également pour établir des liens entre instants et intervalles exprimant qu'un instant est à l'intérieur ou au début d'un intervalle.

Par contre cette ontologie reste très générale quand à la description des intervalles qui peuvent s'avérer un peu particulier. Par exemple rien n'est explicité sur un intervalle qui serait constitué d'un instant, ou sur un intervalle qui aurait un début mais pas de fin (infini) ou inversement.

2. **Relation *Before***
   La relation *before* est définie entre les *TemporalEntity*, et oriente chronologogiquement le temps. Si l'entité temporelle T1 est située avant l'entité temporelle T2, alors la fin de T1 survient avant le début de T2 ; donc une relation *before* va permettre d'indiquer que T1 est avant T2.
   La relation *after* est définie en terme de propriété inverse à la relation *before*.

   Si un instant est à l'intérieur d'un intervalle, alors le début de l'intervalle est avant l'instant t, qui est avant la fin de l'intervalle. Ceci exprime le principe de la propriété *inside*.

3. **Relations entre les Intervalles**
   Allen et Furgerson[1, 2] ont défini un modèle de calcul qui représente les relations temporelles qui se nouent entre deux intervalles. Sept relations sont principalements définies et permettent de poser des interprétations sur des informations temporelles. Les relations entre les intervalles sont définies à partir de leurs points de début et de fin. La notion d'intervalle suppose que tous les intervalles sont "réels", c'est à dire qu'ils possèdent un début et une fin qui diffèrent. L'ontologie définit la classe *ProperInterval* qui caractérise les intervalles propres. OWL-Time fournit les relations entre les intervalles, ainsi que leurs relations inverses illustrées dans la figure 4.11 et leurs définitions illustrées en figure 4.12.

4. **Liaison entre le temps et les événements**
   L'ontologie du temps peut se lier à d'autres modèles de connaissances par le biais de quatre prédicats : *atTime*, *during*, *holds*, et *timeSpan*. Nous supposons dans ce contexte, l'existence d'une autre ontologie portant sur les évènements, soit de haut niveau qui fournit la description générale des événements, ou bien une ontologie d'évènements liée à un domaine spécifique.

   Le terme `éventualité` sera utilisé pour couvrir les événements, les processus, les

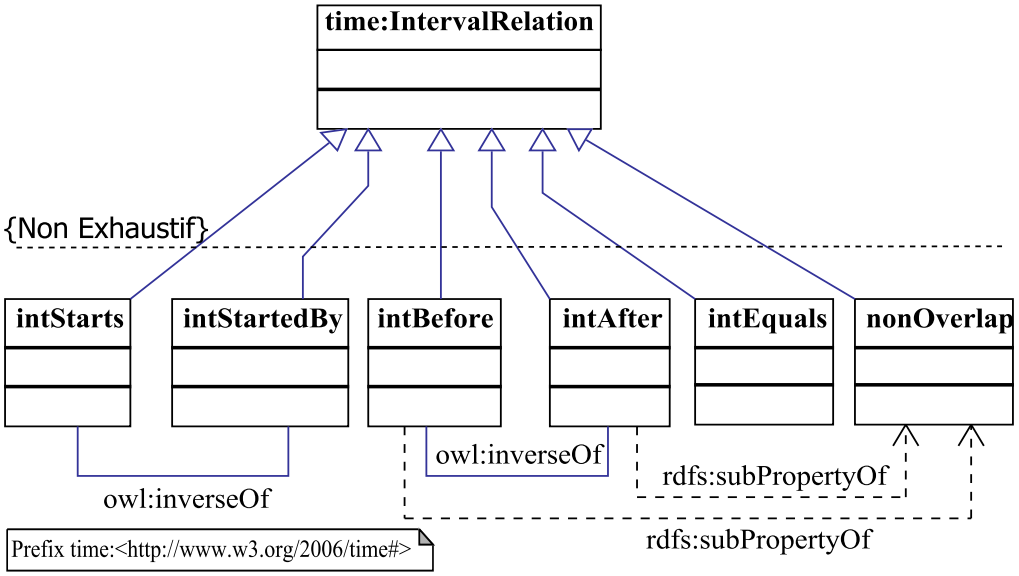

FIGURE 4.11 – Hiérarchie des relations entre les intervalles définie par Allen et Furgerson

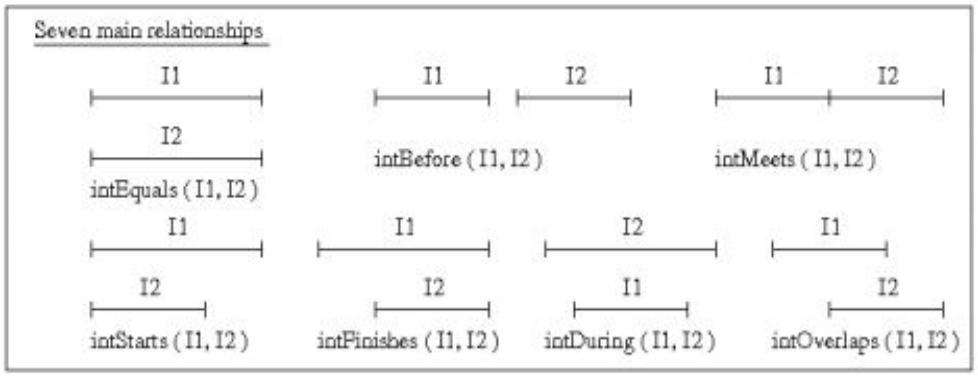

FIGURE 4.12 – Illustration des sept relations principales entre les intervalles

propositions, les états de choses, ou toute autre entité qui peut être située par rapport au temps. Le terme `éventualité` dans cette ontologie est seulement un concept primitif dans l'ontologie du temps.

Le prédicat *atTime* concerne une éventualité à un instant donné, et permet d'indiquer si l'éventualité est passée, en cours ou à prévoir dans le futur.

Le prédicat *during* concerne une éventualité pour un intervalle donné, et permet d'indiquer si l'éventualité est passée, en cours ou à prévoir dans le futur en association dans le cadre de cet intervalle. Si une éventualité s'est déroulée pendant un intervalle, elle est aussi associée aux instants et aux sous-intervalles de l'intervalle.

Souvent, les éventualités dans l'ontologie de l'événement peuvent se représenter par les propositions ; la relation entre ces derniers et le temps est nommé *holds*. Le prédicat *holds* fait partie de l'ontologie de l'événement mais ne fait pas partie de l'ontologie du temps, bien que son second argument puisse être pris en charge par cette dernière.

Le prédicat *timeSpan* concerne les éventualités à des instants ou des intervalles (ou des séquences temporelles des instants et intervalles). L'ajout de ce prédicat à l'ontologie est optionnel.

5. **Différentes façons de la représentation d'une durée d'un événement**
   Il existe différentes façons de représenter les heures et les durées des états et des événements. Dans une première approche, les états et les événements peuvent avoir des durées, et les événements peuvent être des instantanés. Dans une seconde approche, les événements peuvent être instantanés et seuls les états peuvent avoir des durées. Dans une dernière approche, les événements que l'on pourrait considérer comme ayant une durée (par exemple, un trajet maritime) sont modélisés comme un état du système qui est initié et terminé par l'action d'événements instantanés. Dans l'exemple, l'événement instantané est le départ du bâteau, qui correspond au début de l'intervalle et à la transition du système vers un état dans lequel le bâteau progresse sur son parcours. L'état se poursuit jusqu'à ce qu'un autre événement l'arrête et marque la fin de l'intervalle.
   Ces différents points de vue sur les événements sont à définir dans notre appoche méthodologique ; l'ontologie du temps est neutre en ce qui concerne le choix de cette représentation.

6. **La Mesure des intervalles : *Temporal Units***
   Nous détaillons deux approches dédiées à la mesure des intervalles. La première considère les unités de temps comme des fonctions qui vont admettre des intervalles en entrée et en calculer des valeurs réelles en sortie.
   $minutes : Intervals \rightarrow Reals \cup \{Infinity\}$
   Pour exemple avec la fonction minutes : $minutes([5:14, 5:17]) = 3$
   L'autre approche considère que les unités temporelles constituent un ensemble d'entités, appelé *TemporalUnits*, et une seule fonction de mesure du temps est disponible qui prend en entrée l'intervalle et l'unité temporelle et qui retourne des valeurs réelles. $duration : Intervals * TemporalUnits \rightarrow Reals \cup \{Infinity\}$
   Si nous reprenons notre exemple : $duration([\texttt{5:14},\texttt{5:17}], Minute) = 3$, les relations entre les unités temporelles des jours, des mois, et des années sont exprimées dans la modélisation de l'horloge et du calendrier présentée ci-dessous.
   Un jour est envisagé comme un intervalle de calendrier qui commence et finit à minuit. Le deuxième minuit est cependant exclus de l'intervalle. Un jour en tant que durée est cependant un intervalle de 24 heures.
   Tous les moments de la forme `12:xx` heures, y compris `00:00`, font partie de la même heure et du même jour, et tous les moments de la forme `10:15:xx` , comme `10:15:00` , font partie de la même minute.
   Une distinction est faite entre les horloges et les calendriers, car ils diffèrent dans leur façon de numéroter leurs unités d'intervalles.
   Le mois en tant qu'une mesure de durée est lié indirectement à un jour qui est aussi une mesure de la durée, à l'aide du calendrier. Il est ainsi possible de calculer que les mois sont compris entre 28 et 31 jours.

4.2.8 Notre proposition d'extension de l'ontologie du temps

Afin d'introduire la partie temporelle au sein de notre modèle spatio-temporel final, nous définissons dans la figure 4.13 la portion du diagramme OWL-Time (figure 4.10) que nous allons réutiliser par la suite. Nous adaptons le modèle OWL-Time à nos besoins, et de fait, nous avons défini un espace de noms différent qui a pour préfixe `myT`. Les notions `time:TemporalObject`,`time:TemporalEntity`, `time:Instant`, `time:Interval` ont leur définition et leur représentation expliquée dans OWL-Time. Par contre, dans cette extension, la description sur la durée d'une entité qui a été introduite par les divers classes tels que `time:DurationDescription`, `time:CalendarClockDescription`, `time:TemporalUnitType`, illustrées en figure 4.10, sont factorisées dans une classe appelée `myT:TemporalReferenceSystem`. Ce système de référencement du temps fournit un contexte pour le temps d'apparition d'une entité temporelle afin d'établir les relations entre les objets temporels. La relation entre une entité temporelle et son référencement du temps est nommée `myT:hasDurationDescription`.

Nous considérons la durée d'apparition d'une entité, `time:Interval`, modélisée comme étant un `time:ProperInterval` proposé dans OWL-Time. Par conséquence, la relation générale entre les intervalles appelée `hasIntervalRelation` va porter sur la classe `time:Interval`.

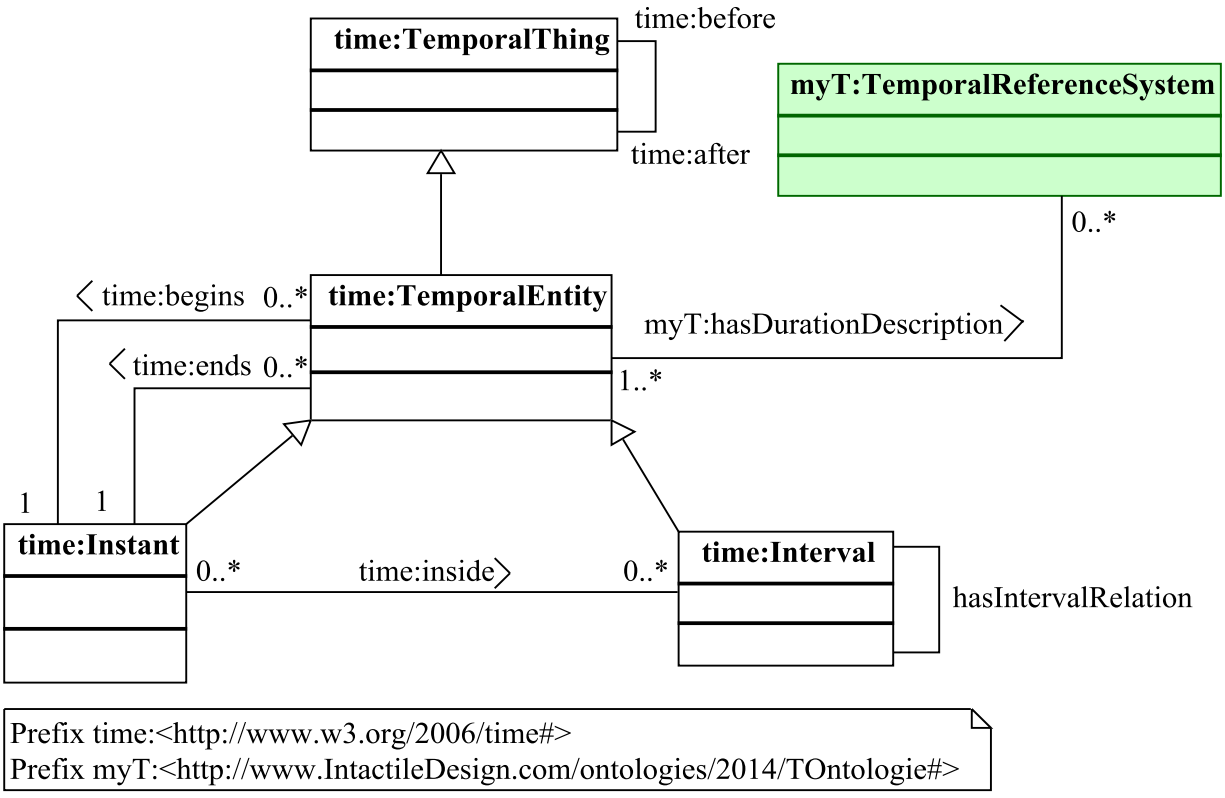

FIGURE 4.13 – Une portion de l'ontologie du temps OWL-Time

4.2.9 Modèle spatio-temporel unifié

L'évolution d'une entité spatiale au cours du temps aboutit à un graphe de relations qui représente les états successifs de cette entité. Ce graphe peut être modélisé à l'aide de modules ontologiques qui réutilisent pour un contexte donné, les ontologies de haut niveau dédiées au temps et à l'espace. Pour la construction des modules ontologiques, nous reprenons le langage de spécification OWL DL (*Ontology Web Language Description Logics*) qui est également le langage choisi pour les ontologies de haut niveau. OWL DL offre un bon compromis avec une expressivité adaptée aux besoins de la modélisation et également des mécanismes d'inférence décidables.

Dans la figure 4.14, nous proposons un modèle conceptuel pour la représentation de l'information spatiale et temporelle d'objets mobiles. Ce modèle tire avantage des entités spatiales du module ontologique spatial proposé en figure 4.8 ainsi que des entités temporelles du module ontologique proposé en figure 4.13 qui explicite les relations temporelles et le déplacement ou l'évolution d'entités au cours du temps. En outre, ce nouveau module ontologique est proposé pour répondre au besoin de modélisation des changements de localisation qui s'opèrent sur les entités spatiales. Le modèle intègre et distingue les relations liées à l'espace (relations topologiques), au temps (relations d'Allen) à la sémantique et à l'identité (relations de filiation).

Les entités mobiles sont subdivisées en parties temporelles appelées `myST:SpaceTimeSlice`. Chacun des *SpaceTimeSlice* est une représentation partielle des entités, valide sur une période ou à un instant de temps donné. Un *SpaceTimeslice* est lié au diagramme spatial et au diagramme temporel à l'aide de quatre liens. La relation `myST:hasGeometry`, entre un `myST:SpaceTimeSlice` et un `geo:Geometry` de GeoSPARQL, une sous propriété de `geo:hasGeometry` de GeoSparql qui définit la géométrie d'un *SpaceTimeSlice* de l'entité mobile. La relation `myST:isTimeSliceOf`, entre un `myST:SpaceTimeSlice` et un `geo:Feature` de modèle GeoSparql, indique à quel *Feature* appartient un *SpaceTimeSlice*. La relation `myST:atTime` et `time:during`, entre `myST:SpaceTimeSlice` et `time:Instant` ou `time:Interval`, expriment la validité d'un *SpaceTimeSlice* à un instant ou à un intervalle de temps donné. La relation `myST:hasFiliation` indique les changements qui s'opèrent au cours du temps entre les *SpaceTimeSlice*. Cette relation est plutôt destinée aux objets mobiles comme `myS:Way` et `myS:Port` qui peuvent avoir une évolution ou modification géométrique dans le temps. Par conséquent, la représentation complète d'une entité au cours du temps est le résultat de l'agrégation de tous les *SpaceTimeSlice*. Par exemple, nous pouvons avoir une trajectoire d'un navire dans une durée du temps grâce à l'ensemble des `myS:MovingPoint` de ce navire.

Grâce à la relation `myST:hasGeometry`, les *SpaceTimeSlice* peuvent disposer des relations spatiales de `geo:Geometry` et par la relation `time:during` ils disposent des relations temporelles définies entre les durées du temps dans le modèle temporel. Ces qualités facilitent la manipulation des entités mobiles au regard du temps et de l'espace.

Un diagramme d'objets en figure 4.15 est également proposé et vient illustrer le diagramme de classes spatio-temporel à partir de données provenant de notre jeu de données en figure 4.1.

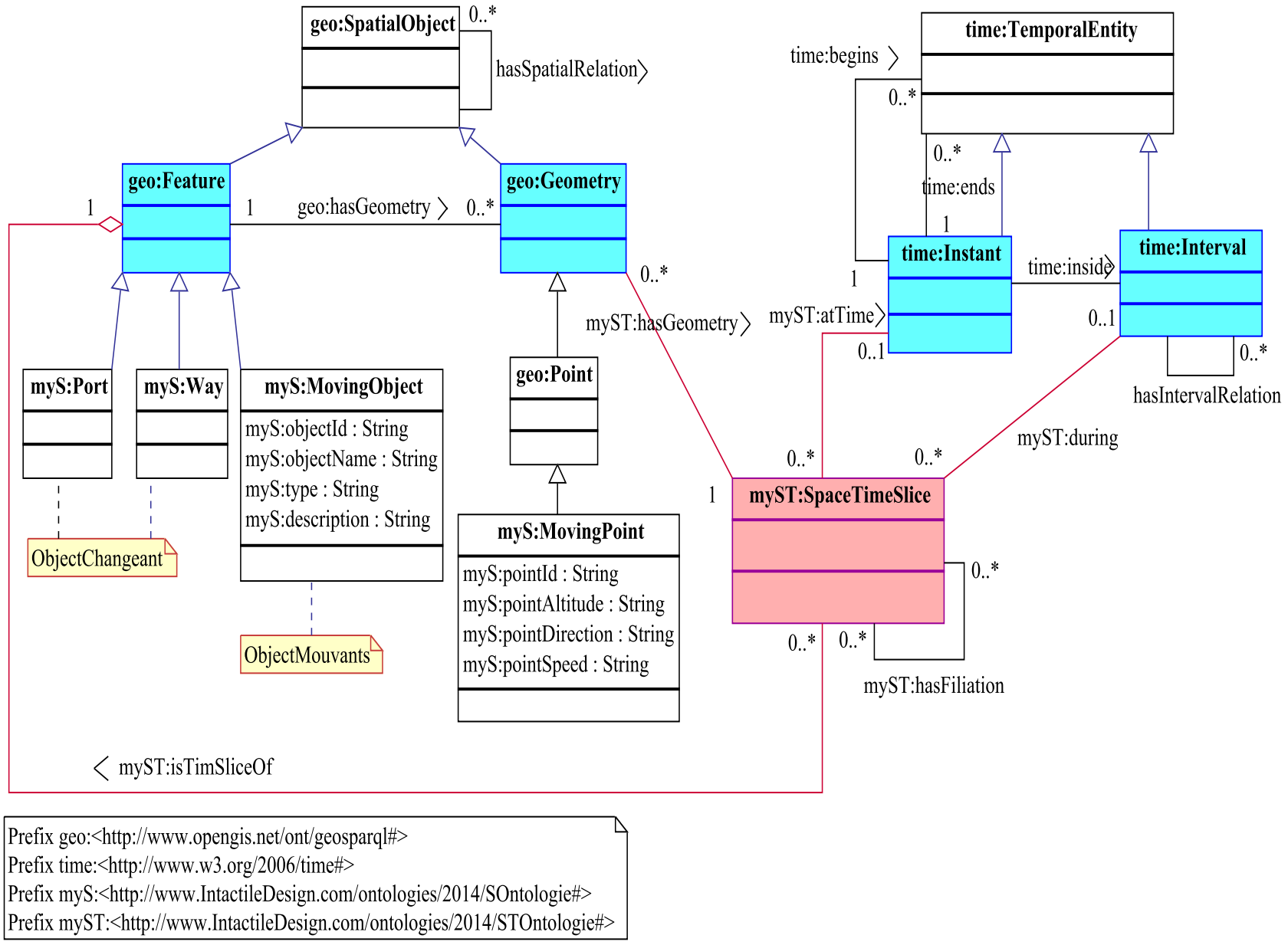

FIGURE 4.14 – Modèle de l'ontologie spatio-temporal

FIGURE 4.15 – Diagramme d'objets

CHAPITRE 5

MISE EN PRATIQUE

5.1 Les systèmes de persistance

L'importance est maintenant donnée à l'implantation de systèmes de persistance qui s'adossent à l'ensemble des modèles de connaissances construits ou réutilisés. Le choix de triplestores se fait naturellement pour ces premiers travaux puisque nous pouvons y implanter directement les schémas OWL qui ont été définis. Le jeu de données est également rendu disponible sous forme d'individus qui se conforment à la définition des modèles.

5.1.1 Triplestore *Parliament*

Parliament est un des triplestores qui intègre des fonctions spatiales provenant des préconisations de GeoSPARQL. Nous l'avons donc choisi à ce titre. Dans une première approche, nous pouvons définir un point d'accès SPARQL ou *EndPoint SPARQL* s'appuyant sur *Parliament*. *Parliament* offre une implémentation qui permet d'indexer les données GeoSPARQL et un moteur de recherche qui prend en charge une partie des mécanismes de requêtage de GeoSPARQL. A cet effet, la définition d'une structure d'index spatial permet de rendre efficace à la fois les requêtes relationnelles spatiales et le stockage des données. Avant la définition récente de GeoSPARQL, Parliament indexait les données s'appuyant sur le modèle GeoOWL et permettait l'interrogation des relations topologiques RCC8 et l'*OGC Simple Features*.
Parliament dispose de diverses fonctionnalités que nous listons ici :

- *Insert Data* pour stocker les donnés sur la base de données Parliament,
- *Query* pour interroger les données insérées dans la base,
- *Explore* pour explorer les classes et les propriétés gérées (l'affichage élémentaire renvoie l'URI de chaque ressource cible),
- *SPARQL/Update* pour mettre à jour les données stockées grâce aux requêtes de mise à jour,
- *Export* pour exporter les graphe de ressources.

Il est aussi possible de communiquer avec le serveur du Parliament par l'API Java associée à Eclipse qui rend ainsi possible l'insertion des données ou l'exécution de requêtes SPARQL/GeoSPARQL grâce à des méthodes stubs (proxy) de l'API.

Nous avons importé les modèles OWL dans Parliament. Le fichier de sérialisation au format RDF/XML est illustré en partie dans le listing 5.1. Le jeu de données a également été inséré avec succès et un premier jeu de requêtes a été testé.

Listing 5.1 – Test

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:p2="http://www.IntactileDesign.com/ontologies/2014/TOntologie.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:p1="http://www.IntactileDesign.com/ontologies/2014/SOntologie.owl#"
    xmlns="http://www.IntactileDesign.com/ontologies/2014/STOntologie#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.IntactileDesign.com/ontologies/2014/STOntologie">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.IntactileDesign.com/ontologies/2014/TOntologie.owl"/>
    <owl:imports rdf:resource="http://www.IntactileDesign.com/ontologies/2014/SOntologie.owl"/>
  </owl:Ontology>
  <owl:Class rdf:ID="TimeSlice"/>
  <owl:ObjectProperty rdf:ID="during">
    <rdfs:domain rdf:resource="#TimeSlice"/>
    <rdfs:range rdf:resource="http://www.IntactileDesign.com/ontologies/2014/TOntologie.owl#Interval"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="atTime">
    <rdfs:range rdf:resource="http://www.IntactileDesign.com/ontologies/2014/TOntologie.owl#Instant"/>
    <rdfs:domain rdf:resource="#TimeSlice"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isTimeSliceOf">
    <rdfs:domain rdf:resource="#TimeSlice"/>
    <rdfs:range rdf:resource="http://www.opengis.net/ont/geosparql#Feature"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasGeometry">
    <rdfs:subPropertyOf rdf:resource="http://www.opengis.net/ont/geosparql#hasGeometry"/>
    <rdfs:domain rdf:resource="#TimeSlice"/>
    <rdfs:range rdf:resource="http://www.opengis.net/ont/geosparql#Geometry"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasFiliation">
    <rdfs:range rdf:resource="#TimeSlice"/>
    <rdfs:domain rdf:resource="#TimeSlice"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

L'exemple suivant crée un navire et un port avec un point et un polygone, en suite nous interrogeons de manière à vérifier si le navire est dans le port.

```
mySx:ship1 a myS:Ship;
rdfs:label "MSC Banu";
geo:hasGeometry mySx:MovingPoint1 .
mySx:MovingPoint1 a geo:Point;
  geo:asWKT "POINT(-77.03524 38.889468)"^^geo-sf:WktLiteral.
mySx:port1 a myS:Port;
rdfs:label "Port Marseille";
geo:hasGeometry mySx:Polygon1 .
mySx:Polygon1 a geo:Polygon;
geo:asWKT
 "POLYGON((-77.05 38.87, -77.02 38.87, -77.02 38.9, -77.05 38.9,\\ 77.05 38.87))"
 ^^geo-sf:WktLiteral.


SELECT ?mp1 ?Polygon1
WHERE {
mySx:ship1 geo:hasGeometry ?mp1.
mySx:port1 geo:hasGeometry ?Polygon1.
?mp1 geo:sfWithin ?Polygon1.
}
```

5.1.2 Triplestore TDB

TDB est un composant de Jena exploité pour le stockage et la manipulation de données au format RDF/RDFS/OWL. TDB permet de construire des bases de données RDF (TDB store) performantes en mode centralisé. Pour assurer le partage d'un TDB store, entre plusieurs applications, il faut utiliser le composant *Fuseki* qui permet d'implanter un point d'accès (endpoint) SPARQL qui va s'adosser à TDB pour la persistance et les mécanismes de concurrence sur les données, et fournir des services sur la base du protocole SPARQL pour la requêtage et la mise à le jour des données. Le moteur de requêtes de Jena nommé ARQ n'exploite pas GeoSPARQL pour les requêtes spatiales, mais propose une extension propriétaire de SPARQL pour prendre en charge des requêtes spatiales élémentaires. Nous avons défini un premier prototype de manière à valider notre modèle. Le prototype est organisé au travers de plusieurs paquetages qui sont :

- Package CsvParser : qui parcourt et analyse les fichiers d'entrée (jeu de données au format Csv) et retourne les informations sérialisées dans une liste des items (Item : est une classe JAVA),
- Package DataModel : contient les classes nécessaire pour les objets existants dans le modèle propre à notre jeu de données.
- Package JenaUtils : contient les classes utilitaires pour notamment l'affichage et la conversion de données en différents formats

Nous exploitons en particulier les classes Jena présentées ci-dessous :

- Une classe appelée `ModelFactory` qui définit notre modèle ontologique *OntModel* composé des *OntClass* et *OntProperty*,
- Une classe appelé `ConvertCSVtoRDF` convertit les données CSV au format RDF adossées le modèle d'ontologie défini dans la classe `ModelFactory`,
- Une classe `TDButils` pour la mise en place et la création d'un TDB store et l'exécution des requêtes SPARQL.

5.2 Ontologie spatio-temporelle et formalisme DL

Nous reprenons ci-dessous une portion de l'ontologie, illustrée en figure 4.14, exprimée en logique de description (DL).
Nous avons choisi de donner une description des entités spatiales d'intérêt dans notre domaine comme `myST:MovingObject` qui vient spécialiser `geo:Feature` et qui constitue un concept important de notre apport sur la représentation du spatio-temporel. `myST:MovingObject` est spécialisé par `myS:Ship` qui ne va avoir que des géométries de type `myST:MovingPoint`.

`geo:<http://www.opengis.net/ont/geosparql\#>`
`myST:<http://www.IntactileDESGN.com/ontologies/2014/STOntologie\#>`
`myS:<http://www.IntactileDESGN.com/ontologies/2014/SOntologie\#>`
`myT:<http://www.IntactileDESGN.com/ontologies/2014/TOntologie\#>`

$$\texttt{myST:Feature} \subseteq \texttt{geo:SpatialObject}$$
$$\texttt{myST:MovingObject} \subseteq \texttt{geo:Feature}$$
$$\texttt{myST:MovingObject} \subseteq \leq 1\texttt{myS:ObjectId.String}$$
$$\texttt{myST:MovingObject} \subseteq \leq 1\texttt{myS:ObjectName.String}$$
$$\texttt{myST:MovingObject} \subseteq \leq 1\texttt{myS:type.String}$$
$$\texttt{myST:MovingObject} \subseteq \leq 1\texttt{myS:description.String}$$

$$\texttt{myST:Point} \subseteq \texttt{geo:Geometry}$$
$$\texttt{myST:MovingPoint} \subseteq \texttt{geo:Point}$$
$$\texttt{myS:Ship} \subseteq \texttt{myST:MovingObject} \cap \forall\texttt{geo:hasGeometry.myS:MovingPoint}$$
$$. \qquad \cap\exists\texttt{geo:hasGeometry.myS:MovingPoint}$$

$$\texttt{time:Instant} \subseteq \texttt{time:TemporalEntity}$$
$$\texttt{time:Interval} \subseteq \texttt{time:TemporalEntity}$$
$$\texttt{time:Instant} \subseteq \forall\texttt{time:inside.time:Interval}$$
$$\texttt{time:TemporalEntity} \subseteq \leq 1\texttt{time:begins.time:Instant}$$
$$\texttt{time:TemporalEntity} \subseteq \leq 1\texttt{time:ends.time:Instant}$$

$$\texttt{myST:TimeSlice} \subseteq \forall\texttt{myST:hasGeometry.geo:Geometry}$$
$$. \qquad \cap\exists\texttt{myST:hasGeometry.geo:Geometry}$$
$$\texttt{myST:TimeSlice} \subseteq \leq 1\texttt{myST:atTime.time:Instant}$$
$$\texttt{myST:TimeSlice} \subseteq \forall i \in \{0,1\}\texttt{myST:during.time:Interval}$$
$$\texttt{myST:TimeSlice} \subseteq \exists\texttt{myST:isTimeSliceOf.geo:Feature}$$
$$. \qquad \cap \leq 1\texttt{geo:isTimeSliceOf.geo:Feature}$$

CONCLUSION ET PERSPECTIVES

Nous partons du constat que les solutions de gestion de données relationnelles tendent à montrer leurs limites en terme de gestion et d'exploitation de grandes masses de données géo-référencées ; et nous avons en conséquence cherché à étudier différents systèmes qui facilitent le stockage, l'interrogation et la modélisation de gros volumes de données potentiellement évolutives et complexes. Pour cela, nous avons défini dans un premier temps un jeu de données spatialisées extrait du modèle OpenStreetMap au format XML. Nous avons ensuite réalisé un état de l'art autour des différents systèmes de gestion de données existants pouvant apporter des éléments de réponse en terme de gestion de grandes masses de données et/ou de gestion de données complexes. Les différentes orientations prises au sein de la mouvance NoSQL ont été explorées. Nous avons expérimenté certains de ces systèmes afin d'en mieux comprendre les fonctionnalités et les modèles sous-jacents pour des données spatialisées. Nous avons abordé partiellement la potentialité de trois systèmes à prendre en charge de manière efficace des données spatialisées. Nous avons réorienté les objectifs du stage en préférant porter notre attention sur la définition d'un modèle de connaissances spatio-temporels qui prend la forme de trois modules ontologiques : module de domaine "objet mobile" spatial, module de domaine "objet mobile" temporel et modèle cadre spatio-temporel. Un premier choix de système de persistance porte sur les triplestores et notamment sur les systèmes Parliament et TDB. La comparaison entre différents systèmes de gestion de données pourrait se faire dans un futur proche en s'appuyant sur les modèles de connaissances construits. La réorientation du travail se justifie par le besoin de disposer d'un modèle spatio-temporel stable avant de construire un large jeu de données d'étude conforme à ce modèle. Le jeu de données doit être suffisamment riche en matière d'expressivité et volumineux de manière à pouvoir conduire des expérimentations significatives, et des comparaisons entre systèmes de gestion de données, porteuses de sens. En ce sens, un autre point à aborder, concerne les calculs et raisonnements spatio-temporels qui peuvent être conduits sur le jeux de données test. Nous avons montré que différents travaux avaient permis de poser des définitions rigoureuses de ce qu'était par exemple une relation topologique de chevauchement entre deux entités spatiales ou bien une relation temporelle de séquentialité entre deux évènements. Il faut maintenant définir un jeu de requêtes pertinent qui rende compte de la diversité des liens qui peuvent associer deux ou plusieurs objets spatio-temporels entre eux et qui soit en particulier adapté à la demande qui nous est faite de détecter des comportements inhabituels. Un point d'intérêt peut par exemple être essayer d'évaluer l'apport des triplestores, des ontologies et des mécanismes de raisonnement face aux systèmes d'information géographiques qui intègrent bon nombre des fonctions de calcul spatial.

RÉFÉRENCES DES SYSTÈMES

Afin de proposer des informations d'ordre pratique sur les différents systèmes que nous avons regardé, nous proposons un tableau 5.1 qui récapitule pour chaque système l'adresse Web de son site et le modèle de sa licence d'exploitation (parmi propriétaire, Apache, BSD et GPL). Les systèmes sont listés par ordre alphabétique.

Système	Licence	Le site web pour plus d'info
Cassandra	Apache	incubator.apache.org/cassandra
Clustrix	Prop	clustrix.com
CouchDB	Apache	couchdb.apache.org
HBase	Apache	hbase.apache.org
MongoDB	GPL	mongodb.org
MySQL Cluster	GPL	mysql.com/cluster
Neo4j	AGPL	neo4j.org
Redis	BSD	code.google.com/p/redis
Riak	Apache	riak.basho.com
Scalaris	Apache	code.google.com/p/scalaris
ScaleDB	GPL	scaledb.com
SimpleDB	Prop	amazon.com/simpledb
Voldemort	None	project-voldemort.com
VoltDB	GPL	voltdb.com

FIGURE 5.1 – Références des principaux systèmes explorés

BIBLIOGRAPHIE

[1] J. F. Allen. Towards a general theory of action and time. *Artificial intelligence*, 23(2) :123–154, 1984.

[2] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of logic and computation*, 4(5) :531–579, 1994.

[3] R. ANGLES. A comparison of current graph database models. 2012.

[4] P. ATZENI, F. BUGIOTTI, and L. ROSSI. Uniform access to nosql systems. 2013.

[5] S. Auer, J. Lehmann, and S. Hellmann. Linkedgeodata : Adding a spatial dimension to the web of data. In A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 731–746. Springer Berlin Heidelberg, 2009.

[6] R. Battle and D. Kolas. Enabling the geospatial semantic web with parliament and geosparql. *Semantic Web*, 3(4) :355–370, 2012.

[7] R. Cattell. Scalable sql and nosql data stores. 2010.

[8] 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. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, 2006.

[9] E. Clementini and R. Laurini. Un cadre conceptuel pour modéliser les relations spatiales. *Revue des Nouvelles Technologies de l'Information (RNTI)*, 14 :1–17, 2008.

[10] S. Coast. How openstreetmap is changing the world. In K. Tanaka, P. Fröhlich, and K.-S. Kim, editors, *Web and Wireless Geographical Information Systems*, volume 6574 of *Lecture Notes in Computer Science*, pages 4–4. Springer Berlin Heidelberg, 2011.

[11] E. F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6) :377–387, 1970.

[12] D. Damen and T. Van den Bulcke. Towards a flexible semantic framework for clinical trial eligibility using topic maps. In *Proceedings of the ACM SIGKDD Workshop on Health Informatics, HI-KDD'12*, 2012.

[13] E. DAN PRITCHETT. Base,un acid alternative. 2008.

[14] B. Harbelot, H. Arenas, and C. Cruz. un modèle sémantique spatio-temporel pour capturer la dynamique des environnements. In *Fouille de données spatiales et temporelles*, page 39 à 54, Rennes, France, Jan. 2014.

[15] B. Harbelot, H. Arenas, C. Cruz, et al. Un modèle sémantique spatio-temporel pour capturer la dynamique des environnements. *Fouille de données spatiales et temporelles*, 2014.

[16] B. Haslhofer, E. Momeni Roochi, B. Schandl, and S. Zander. Europeana rdf store report. 2011.

[17] J. Herring. Opengis implementation standard for geographic information-simple feature access-part 1 : Common architecture. *OGC Document*, 2011.

[18] J. R. Hobbs and F. Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(1) :66–85, 2004.

[19] O. G. C. Inc. *OpenGIS Implementation Standard for Geographic information - Simple feature access*. 1.2.1 edition, 2011.

[20] A. Kralidis. Geospatial open source and open standards convergences. In G. Hall and M. Leahy, editors, *Open Source Approaches in Spatial Data Handling*, volume 2 of *Advances in Geographic Information Science*, pages 1–20. Springer Berlin Heidelberg, 2008.

[21] C.-T. Lu, J. Santos, RaimundoF., L. Sripada, and Y. Kou. Advances in gml for geospatial applications. *GeoInformatica*, 11(1) :131–157, 2007.

[22] M. Lupp. Open geospatial consortium. In *Encyclopedia of GIS*, pages 815–815. Springer, 2008.

[23] H. NAACKE and A. DOUCET. Optimisation des bases de données. *Techniques de l'ingénieur Bases de données*, base documentaire : TIB309DUO.(ref. article : h3702), 2014. fre.

[24] A. POLLERES, M. ARENAS, P. KRONER, and P. PATEL-SCHNEIDER. *Reasoning Web*. 7th International Summer School, 2011.

[25] K. Rohloff, M. Dean, I. Emmons, D. Ryder, and J. Sumner. An evaluation of triple-store technologies for large data stores. In *On the Move to Meaningful Internet Systems 2007 : OTM 2007 Workshops*, pages 1105–1114. Springer, 2007.

[26] O. Teste, G. Pujolle, and F. Ravat. Bases de données relationnelles. *Techniques de l'ingénieur Bases de données*, base documentaire : TIB309DUO.(ref. article : h3860), 2014.

ANNEXES

Hiérarchie des relations spatiales en langage OWL DL

$geo : spatialRelation \subseteq geo : hasRelationSpatial$
$geo : GeometryTopologyExtension \subseteq geo : spatialRelation$

$geo : sfQueryFunction \subseteq geo : GeometryTopologyExtension$
$geof : sfContains \subseteq geo : sfQueryFunction$
$geof : sfCrosses \subseteq geo : sfQueryFunction$
$geof : sfDisjoint \subseteq geo : sfQueryFunction$
$geof : sfEquals \subseteq geo : sfQueryFunction$
$geof : sfIntersects \subseteq geo : sfQueryFunction$
$geof : sfOverlaps \subseteq geo : sfQueryFunction$
$geof : relate \subseteq geo : sfQueryFunction$
$geof : sfTouches \subseteq geo : sfQueryFunction$
$geof : sfWithin \subseteq geo : sfQueryFunction$

$geo : EHQueryFunction \subseteq geo : GeometryTopologyExtension$
$geof : ehEquals \subseteq geo : EHQueryFunction$
$geof : ehDisjoint \subseteq geo : EHQueryFunction$
$geof : ehMeet \subseteq geo : EHQueryFunction$
$geof : ehOverlap \subseteq geo : EHQueryFunction$
$geof : ehCovers \subseteq geo : EHQueryFunction$
$geof : ehCoveredBy \subseteq geo : EHQueryFunction$
$geof : ehInside \subseteq geo : EHQueryFunction$
$geof : ehContains \subseteq geo : EHQueryFunction$

$geo : RCC8QueryFunction \subseteq geo : GeometryTopologyExtension$
$geof : rcc8eq \subseteq geo : RCC8QueryFunction$
$geof : rcc8dc \subseteq geo : RCC8QueryFunction$
$geof : rcc8ec \subseteq geo : RCC8QueryFunction$
$geof : rcc8po \subseteq geo : RCC8QueryFunction$
$geof : rcc8tppi \subseteq geo : RCC8QueryFunction$
$geof : rcc8tpp \subseteq geo : RCC8QueryFunction$
$geof : rcc8ntpp \subseteq geo : RCC8QueryFunction$
$geof : rcc8ntppi \subseteq geo : RCC8QueryFunction$

$my : ProccesingFunction \subseteq geo : spatialRelation$
$geof : boundary \subseteq my : ProccesingFunction$
$geof : buffer \subseteq my : ProccesingFunction$
$geof : convexHull \subseteq my : ProccesingFunction$

$geof : difference \subseteq my : ProccesingFunction$
$geof : envelope \subseteq my : ProccesingFunction$
$geof : symDifference \subseteq my : ProccesingFunction$
$geof : union \subseteq my : ProccesingFunction$
$geof : intersection \subseteq my : ProccesingFunction$

$my : MeasurementFunction \subseteq geo : spatialRelation$
$geo : coordinateDimension \subseteq my : MeasurementFunction$
$geo : dimension \subseteq my : MeasurementFunction$
$geof : distance \subseteq my : MeasurementFunction$
$geof : getSRID \subseteq my : MeasurementFunction$
$geo : isEmpty \subseteq my : MeasurementFunction$
$geof : symDifference \subseteq my : MeasurementFunction$
$geof : union \subseteq my : MeasurementFunction$

Hiérarchie des relations intervalles en langage OWL DL

$time : intervalRelation \subseteq time : hasIntervalRelation$
$time : intEquals \subseteq time : intervalRelation$
$time : intBefore \subseteq time : intervalRelation$
$time : intMeets \subseteq time : intervalRelation$
$time : intOverlaps \subseteq time : intervalRelation$
$time : intStarts \subseteq time : intervalRelation$
$time : intDuring \subseteq time : intervalRelationn$
$time : intFinishes \subseteq time : intervalRelation$
$time : nonoverlap \subseteq time : intervalRelation$
$time : startsOrDuring \subseteq time : intervalRelation$
$time : intStarts \subseteq time : startsOrDuring$
$time : intDuring \subseteq time : startsOrDuring$
$time : intBefore \subseteq time : before$

$time : inverseIntervalRelation \subseteq time : hasIntervalRelation$
$time : intAfter \equiv time : intBefore^{-}$
$time : intMetBy \equiv time : intMeets^{-}$
$time : intOverlappedBy \equiv time : intOverlaps^{-}$
$time : intStartedBy \equiv time : intStarts^{-}$
$time : intContains \equiv time : intDuring^{-}$
$time : intFinishedBy \equiv time : intFinishes^{-}$