Méthodologie de génération d'applications auto-adaptatives grâce à un cadre de développement dirigé par les ontologies

par

Louis BHÉRER

THÈSE PRÉSENTÉE À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DU
DOCTORAT EN GÉNIE
Ph.D.

MONTRÉAL, LE 2 MARS 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CETTE THÈSE A ÉTÉ ÉVALUÉE

PAR UN JURY COMPOSÉ DE :

M. Christian Desrosiers, directeur de thèse
Département de Génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Luc Vouligny, codirecteur de thèse
Unité Science des données et calcul haute performance à l'Institut de recherche d'Hydro-Québec

M. Marc Paquet, président du jury
Département de génie des systèmes à l'École de technologie supérieure

Mme Sylvie Ratté, professeur
Département de Génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Petko Valtchev, professeur
Département d'informatique à l'Université du Québec à Montréal

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 21 DÉCEMBRE 2020

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

La présente thèse est le fruit de cinq années de travail dans les laboratoires de l'Institut de recherche d'Hydro-Québec. Pendant cette période, sept autres stagiaires de niveau baccalauréat ont été embauchés pour travailler sur mes projets afin de m'aider à les porter à terme. Plusieurs stagiaires étant restés plus d'une session, cela représente 12 sessions de travail.

J'aimerais donc vous remercier pour tout votre travail et les bons moments passés ensemble. Merci à vous sept : Billel Redouane, Aristide Mekontso, Mohammed Djoufelkit, Saloua Djebeli, Youssef Rhindi, Guillaume Larente et Gabriel Beaudoin.

J'aimerais aussi remercier le CRSNG, le FRQNT, l'ÉTS et Hydro-Québec qui m'ont apporté le soutien financier, matériel et professionnel nécessaire tout au long de ce doctorat et qui ont rendu possible la réalisation de ce projet.

De plus, j'aimerais remercier les directeur et co-directeur de cette recherche, Christian Desrosiers et Luc Vouligny dont les conseils et les encouragements m'ont permis de mener à bien cette recherche.

Owl Monkey, la troisième application développée dans le cadre de cette recherche, représente à elle seule plus de trois ans de travail. Elle a été conçue, développée et mise en production par les stagiaires susmentionnés et moi-même grâce à la précieuse aide des chercheurs des unités de Science des données et de Science des matériaux, des membres de l'unité des Systèmes informationnels scientifiques ainsi que des membres de plusieurs autres unités de l'IREQ, des membres de la Vice-présidence des technologies de l'information, des gens des unités d'affaires, des gestionnaires et du personnel de soutien. Merci à vous tous!

J'aimerais finalement remercier tous les gens qui m'ont aidé dans mes objectifs mais aussi tous ceux que j'ai côtoyé pendant ce stage. C'est grâce à vous si mon séjour à l'IREQ et à Hydro-Québec fut si agréable et je vous en suis très reconnaissant.

Merci à : Arnaud Zinflou, Claude Lafond, Olfa Ben Sik Ali, Alandre Magloire, Stéphane Alari, Badoung Ding, Line Gaulin, Linda, Mireille et Carlos, Étienne Martin, Eric Ouellet, Danny Ouellet, Jean-François Losier, Alain Côté, Jean-François Boudreau, Olivier Blancke, Michaël Tessier, Sharon Mandair, Bouchra Tenni, Olivier Beaudoin, Dérick Houde, Antoine Côté,

Jonathan Picher, Maxime Turmel, Michel Héon, Xiaohui Luo, Alexandre Bouffard, Mathieu Viau, Christian Langheit, Michel Gauvin, Nathalie DeGuise, Louis-Alexandre Leclaire, François Mirallès, Guillaume Houle, Ribal Atallah, Alexia Marchand, France Guillemette, Denis Valiquette, Patrick Jeandroz, Régis Houde, Raouf Naggar, Marie-Lise Tremblay, Michel Perrier, Olivier Kokoko, Mohamed Shoul, Chuma Francis Mugombozi, Andrea Paolella, Charline David, Fabienne Chastelas, Line Laniel, Renée St-Germain, Josée Brodeur, Louis Lamarche, André Tarass, Jonathan Racine, Jacquelin Bisson, Sylvain Sergerie, Daniel Marcil, Yannick Barrolet, François Léonard, Alain Forcione, Éric Truchon, France Trudel, Anne-Marie Giroux, Lionel Reynaud, Sébastien Poirier, Cécile Escaich, Gabrielle Turcot, Mireille Massala-Kivoua, Michel Duval, Martha Cea, Kevin Kombate, Jérôme Bélanger, Jean-François Richard, Philippe Jodoin, Carlos Vasquez, Danny Lok, Jacques Bhérer, Cherif Belkalem, Mohamed Hacene Medjadji, Steve Valois, Atieh Delavari, Patrice Brunelle, Dominique Tapsoba, Claude Hudon, Normand Amyot, Emmanuel Bigeon, Stéphane Godin, Laetitia Maugain, Minh Au, Redouane Mejdi, Hind Dirani, Amélie Combette, Anne Flaus, Pascal Feo, Philippe Nichols, Meryem Benghanem, Rock Beaudoin, Samuel Cupillard, Dragan Komljenovic, Jocelyn Jalbert, Robert DuMais, Pierre Couture, Lucie Bibeau, Julien Beaudry, Marthe Kassouf, Mohamed Gaha, Mouhamadou Makhtar Dione, Champlain Landry, Jonathan Hamelin, Nelly Sénéchal, Hatime Enhas, Louis Gastonguay, Martin-Pierre Dumouchel, Sébastien Favron, Sylvain Morin, Danielle De Sève, Mariela Rodriguez, Mylène Mastrostefano, Augustine Eteme Dame, Luc Cauchon, Sébastien Bergeron Côté, Oscar Arroyo, Océane Chotard, Maurice Tardif, Francisc Zavoda, Éric Villemure, Louis Lépine, Basile Agba, Christian Bélanger, Violaine Dorval, Nadine Ibrahim, Amira Dems, Patrick Picher, Alexandre Gilbert, Gaétan Lantagne, Jean Matte, Jérôme Gosset et tous les autres.

J’aimerais finalement remercier du fond du cœur mes parents Andrée et Daniel, mon frère Mathieu et sa famille, ainsi que ma copine, Martine, pour le soutient et les encouragements.

Méthodologie de génération d'applications auto-adaptatives grâce à un cadre de développement dirigé par les ontologies

Louis BHÉRER

RÉSUMÉ

Cette thèse présente trois applications qui ont été conçues, développées et déployées dans des cas d'utilisation réels pour Hydro-Québec, le producteur, transporteur et fournisseur d'électricité provincial du Québec. Ces applications ont été créées de façon à présenter des propriétés d'auto-adaptabilité à l'évolution des connaissances de l'entreprise. L'objectif de cette recherche appliquée était d'évaluer la faisabilité de l'utilisation des technologies du Web sémantique pour créer de telles applications Web qui soient utilisables par des utilisateurs débutants.

Le développement de ces trois applications a entrainé des contributions propres à chacune d'elle. La première application est un navigateur sémantique textuel et elle a permis de démontrer qu'il était possible d'atteindre l'indépendance conceptuelle grâce aux technologies du Web sémantique. La deuxième application est un constructeur de requêtes visuelles SPARQL. Son développement a principalement permis d'explorer la transposition des fonctionnalités associées à la construction de requêtes SQL au langage SPARQL. La troisième application est un constructeur de vues tirant profit des leçons apprises lors de la réalisation des deux applications précédentes. Les contributions associées à cette application comprennent : 1. Une composante logicielle inédite d'exploration ontologique; 2. Une liste de recommandations de modélisation ontologique permettant aux modèles d'être utilisés par des applications adoptant cette méthodologie; 3. Une ontologie de visualisation qui est une première tentative vers la création d'un standard afin de guider et personnaliser les interfaces utilisateurs par des annotations faites à même les ontologies de domaine; 4. Un jeu de pré-inférence permettant la réconciliation du concept d'héritage dans les paradigmes orienté-objets avec le concept de classification du paradigme de la logique de premier ordre; et 5. Une approche permettant de croiser des données d'ontologies OWL de manière *ad hoc*.

Les expérimentations présentées dans cette thèse ont permis de développer une méthodologie pour générer des applications auto-adaptatives manœuvrables par des néophytes, comblant ainsi un manque de la littérature. La faisabilité d'une telle approche a été prouvée par la mise en production de la troisième application, générée grâce à cette méthodologie, dans les systèmes de l'entreprise. Cette recherche permet de croire que de telles applications pourront entraîner une réduction des coûts de développement et de maintenance des logiciels. De plus, la généricité et l'auto-adaptabilité des outils développés font de cette approche une candidate de choix pour le développement des applications de l'Industrie 4.0.

Mots-clés: application auto-adaptative, génération d'applications, Web sémantique, RDF, OWL, Industrie 4.0, ingénierie dirigée par les ontologies.

Méthodologie de génération d'applications auto-adaptatives grâce à un cadre de développement dirigé par les ontologies

Louis BHÉRER

ABSTRACT

This thesis presents three applications that have been designed, developed and deployed in real use cases for Hydro-Quebec, the Quebec provincial producer, carrier and supplier of electricity. These applications were created in such a way as to exhibit properties of self-adaptability to the evolution of knowledge of the company. The objective of this applied research was to assess the feasibility of using semantic web technologies to create such web applications that are usable by novice users.

The development of these three applications resulted in contributions specific to each of them. The first application is a textual semantic browser and it has demonstrated that it is possible to achieve conceptual independence thanks to semantic web technologies. The second application is a SPARQL visual query builder. Its development has mainly enabled the exploration of the transposition of functionalities associated with the construction of SQL queries to the SPARQL language. The third application is a view builder taking advantage of the lessons learned during the realization of the two previous applications. The contributions associated with this application include: 1. A unique software component for ontological exploration; 2. A list of ontological modeling recommendations allowing models to be used by applications applying this methodology; 3. A visualization ontology which is a first attempt towards the creation of a standard in order to guide and personalize user interfaces by annotations made within domain ontologies; 4. A set of pre-inferences allowing reconciliation of the concept of inheritance in object-oriented paradigms with the concept of classification of the paradigm of first-order logic; and 5. An approach for crossing data from OWL ontologies in an *ad hoc* manner.

The experiments presented in this thesis made it possible to develop a methodology to generate self-adaptive applications maneuverable by neophytes, thus filling a gap in the literature. The feasibility of such an approach has been proven by the release of the third application, generated using this methodology, into the company's systems. This research suggests that such applications could lead to reduced software development and maintenance costs. In addition, the genericity and self-adaptability of the tools developed make this approach a prime candidate for the development of Industry 4.0 applications.

Keywords: self-adaptive application, application generation, semantic web, RDF, OWL, Industry 4.0, ontology-driven engineering.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES REQUÊTES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ABox	*Assertions Box*
ADM	Architecture dirigée par les modèles
API	*Application Programming Interface*
BDR	Base de données relationnelle
BDT	Base de données de triplets
CRUD	*Create, Read, Update, Delete*
CRV	Constructeur de requêtes visuelles
CRV-HQ	Constructeur de requêtes visuelles SPARQL d'Hydro-Québec
CSV	*Comma-Separated Values*
DATE	Données d'analyse des transformateurs électriques
EBox	*Enumeration Box*
ETL	*Extract*, *Transform*, *Load*
EVA	Étude du vieillissement de l'appareillage
GTA	Groupes turbines-alternateurs
HQ	Hydro-Québec
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDM	Ingénierie dirigée par les modèles
IDO	Ingénierie dirigée par les ontologies
IEC/CIM	*International Electrotechnical Commission / Common Information Model*
IIoT	*Industrial Internet of Things*
IoT	*Internet of Things*
IREQ	Institut de recherche d'Hydro-Québec
JDBC	*Java Database Connectivity*
JSON	*JavaScript Object Notation*
JSON-LD	*JSON for Linked Data*
MDI	Modélisation dynamique de l'information
OMG	*Object Management Group*

OO	Orienté-objet
OV	Ontologie de visualisation
OWL	*Web Ontology Language*
PDF	*Portable Document Format*
PL/SQL	*Procedural Language / Structured Query Language*
PNG	*Portable Network Graphics*
PPAL	Performance des paliers autolubrifiants
RD	Récupération de données
RDF	*Resource Description Framework*
REST	*REpresentational State Transfer*
RI	Récupération d'information
SBox	*Save Box*
SCADA	Système de contrôle et d'acquisition de données
SCSC	Système de commande et de surveillance en centrale
SGBD	Système de gestion de base de données
SIA	Système d'information adaptatif
SPARQL	*SPARQL Protocol And RDF Query Langage*
SPIN	*SPARQL Inferencing Notation*
SQL	*Structured Query Language*
SVG	*Scalable Vector Graphics*
SWoT	*Semanctic Web of Things*
SWP	*SPARQL Web Pages*
TBox	*Terminology Box*
UI	*User Interface*
URI	*Uniform Resource Identifier*
VBA	*Visual Basic for Applications*
W3C	*World Wide Web Consortium*
WIMP	*Windows, Icons, Mouse, Pointer*
WoT	*Web of Things*

WS	Web Sémantique
XML	*Extensible Markup Language*
XSD	*XML Schema Definition*

INTRODUCTION

À l'aube de la quatrième révolution industrielle, les nouvelles façons de faire entraînent un besoin de changement dans les approches de conception des applications informatiques soutenant les systèmes d'information.

Dans le paradigme de l'Industrie 4.0, alors que les données provenant de tous les maillons de la chaîne de valeur deviennent interconnectées et disponibles de façon transparente, les technologies du Web sémantique (WS) (Semantic Web, 2020) sont de plus en plus à l'étude comme moyen d'harnacher la connaissance qui y est contenue (Vogel-Heuser & Hess, 2016). En effet, ses capacités de raisonnement, d'interopérabilité et d'auto-adaptativité font du WS une pile technologique de choix pour aborder les défis des systèmes cyber-physiques.

Afin d'atteindre son plein potentiel, l'Industrie 4.0 cherche, entre autres, à faciliter le déploiement rapide d'applications, à développer une approche holistique de conception permettant la réingénierie avec un minimum d'effort et à faciliter l'utilisation de ces applications par des utilisateurs néophytes des technologies sous-jacentes (Thuluva, Anicic, & Rudolph, 2017).

Dans cette optique, cette thèse présente trois applications industrielles des technologies du WS développées pour Hydro-Québec (HQ) (Hydro-Québec, 2020), producteur, transporteur et fournisseur d'électricité provincial du Québec. Elles sont issues des recherches effectuées par son Institut de recherche (IREQ) (IREQ, 2020) sur la gestion des connaissances et le prototypage évolutif (Vouligny & Robert, 2005; Vouligny, Hudon, & Nguyen, 2009; Zinflou, *et al.*, 2013; Gaha, *et al.*, 2013; Gaha, *et al.*, 2015).

Ces trois applications sont, respectivement, un navigateur sémantique, un constructeur de requêtes visuelles sémantique et un constructeur de vues sémantique. Elles ont toutes trois été conçues pour mettre à profit les propriétés d'autoadaptation des technologies du WS. Elles répondent à trois besoins d'affaires distincts de l'entreprise et ont donc fait l'objet de trois projets indépendants. Les connaissances acquises pendant la réalisation des deux premières applications ont permis de proposer une cadre de développement d'applications auto-adaptatives qui a été utilisé pour générer la troisième application.

Ce cadre permet la génération d'applications ayant des capacités d'auto-adaptabilité aux changements de leur modèle de données. Ainsi, les modifications apportées aux ontologies soutenant les informations sont immédiatement visibles pour tous les utilisateurs, sans nécessiter de reprogrammation ou de recompilation de l'application. Cette auto-adaptabilité est particulièrement utile dans un contexte de prototypage évolutif, d'approche agile ou de changements constants à la chaîne de valeurs.

0.1 Problématique

Les entreprises de nos jours se servent quotidiennement des systèmes d'information. « Un système d'information est un ensemble organisé de ressources, lesquelles peuvent être des personnes, des données, des activités ou des ressources matérielles en général. Ces ressources interagissent entre elles pour traiter l'information et la diffuser de façon adéquate en fonction des objectifs d'une organisation (Définition de système d'information, 2020). » Parmi les composantes informatiques des systèmes d'information, on retrouve les applications logicielles qui, sous l'architecture trois-tiers (Architecture trois tiers, 2020), sont constituées d'un interface client, d'un serveur de traitement et d'une base de données.

La plupart des applications actuelles sont constituées de bases de données relationnelles (BDR). Celles-ci jouissent d'une très forte maturité technologique et sont très performantes. De plus, une pléthore de librairies et de cadres de développement sont disponibles pour faciliter l'intégration et le développement d'applications à partir de telles bases de données. Cependant, d'autres types de bases de données ont émergés sur les marchés depuis plusieurs années, tels que les bases de données hiérarchiques, les bases NoSQL, etc., offrant des fonctionnalités différentes que celles des bases de données conventionnelles. Parmi celles-ci se trouvent les bases de données de triplets (BDT), aussi appelées bases de données sémantiques. Ces bases de données développées depuis plusieurs années ont comme principale caractéristique de permettre l'inférence. En effet, leur architecture en graphe jumelée à des standards de modélisation permet à des moteurs d'inférences de déduire de l'information implicite contenue dans les données. Ces déductions sont faites à partir des axiomes de la logique de premier ordre aussi connue sous le nom de calcul de prédicats.

Là n'est pas la seule spécificité de cette technologie. Parmi les autres caractéristiques de ces bases de données se retrouvent : des relations explicitement définies permettant leur compréhension par les machines sans intervention humaine, des possibilités de modélisation dans un monde ouvert et des modèles de données codés dans le même langage que les données. C'est cette dernière caractéristique, permettant une grande flexibilité quant à leur évolution, qui est à la base de la présente recherche.

Les modèles de données des BDR sont codés dans un langage propre aux différentes technologies qui les implémentent. Il est possible de faire des requêtes sur ces modèles, mais cela est fastidieux et s'effectue différemment selon le système de gestion de base de données (SGBD). Dans le cas des BDT, puisque les méta-modèles de données, les modèles de données et les données sont codés dans le même langage, il est possible d'interroger ces trois ensembles simultanément. Cela rend possible l'écriture de requêtes génériques pouvant interroger n'importe quel modèle et jeu de données sans savoir a priori ce qui s'y trouve. Ces requêtes génériques, si elles sont appuyées par des fonctions génériques dans la partie serveur et par des composantes génériques dans l'interface client, peuvent mener à des applications génériques.

Pour les entreprises, les systèmes d'information sont essentiels. Ils constituent le système nerveux par lequel l'information circule, permettant à ses divers organes de réagir aux changements. Lorsque les connaissances d'une entreprise évoluent, il est essentiel de faire évoluer l'organisation en conséquence.

Les applications contenues dans les systèmes d'information actuels ne sont pas particulièrement faciles à faire évoluer. Le savoir sur lequel elles s'appuient peut évoluer sur plusieurs niveaux, impliquant des changements dans les diverses couches de ces applications. Pour que ces changements soient appliqués et que les applications soient ajustées et opérationnelles, cela prend des ressources et du temps. Afin d'effectuer un changement dans un modèle de données, on devra tout d'abord procéder à une étude d'impact sur toutes les applications se servant de ses données puis changer les interfaces clients et les parties serveurs en conséquence. On pourra ensuite procéder aux modifications du modèle de données. Dans une grande entreprise, cela signifie de coordonner le travail de plusieurs employés dans plusieurs départements, augmentant la durée d'attente avant l'obtention d'une nouvelle version

de l'application. Tous ces coûts amènent aussi à n'effectuer que les changements dont le gain dépassera le coût de l'évolution, ou à attendre d'avoir assez de changements et d'effectuer ceux-ci par lots, diminuant la réactivité de l'entreprise.

Des applications capables de s'adapter automatiquement à de tels changements seraient donc souhaitables pour diminuer les coûts de maintenance et d'évolution des systèmes informatiques en entreprise. Elles permettraient de diminuer le temps entre une évolution des connaissances de l'organisation et l'implémentation de celles-ci dans les processus de l'entreprise. En outre, elles permettraient aussi de diminuer le temps de développement requis pour en programmer de nouvelles. En effet, les capacités d'auto-adaptativité de ces applications impliquent qu'elles s'adaptent aux changements survenant sur leur modèle de données. Rien n'empêche alors que l'on change ce modèle de données pour un modèle complètement différent, décrivant un autre domaine de savoir. On peut ainsi fabriquer de nouvelles applications à partir des mêmes composantes logicielles, sans en modifier le code.

Le but de la présente recherche est donc non seulement de construire des applications auto-adaptatives mais de concevoir une méthodologie pour générer de telles applications.

Certaines solutions à la génération automatique d'applications ont d'ores et déjà été proposées dans le cadre de recherches ou de logiciels commerciaux. Celles-ci seront discutées plus en détails dans la revue de littérature. La solution proposée ici se distingue des solutions existantes selon deux perspectives. La première concerne l'auto-adaptivité : la majorité des solutions existantes ne génèrent pas des applications auto-adaptatives. Dans ce cas, les changements apportés au modèle après la génération de l'application doivent être accompagnés d'une nouvelle génération de l'application pour être mis en production et divers problèmes, e.g. de compatibilité des sauvegardes de données entre les versions, peuvent survenir. La deuxième perspective concerne l'usabilité des applications créées. Les quelques solutions existantes générant des applications présentant un certain niveau d'auto-adaptabilité sont conçues pour être manœuvrées par des experts en technologie de l'information. La solution qui sera présentée ici tentera de démocratiser l'usage des technologies sémantiques en utilisant des interfaces intuitives et faciles à utiliser, même par des néophytes des technologies informatiques mises en œuvre.

La généricité de telles applications permet aussi de faire des changements à l'ontologie qui se transmettent aux applications en production sans les affecter négativement. Cela a pour effet de réduire les coûts de développement et de maintenance des applications.

La suite de cette thèse s'orchestre comme suit. Un état des connaissances permettra tout d'abord de faire un survol des notions nécessaires pour pouvoir apprécier les nombreux concepts utilisés tout au long du document.

Le CHAPITRE 3 décriera un navigateur sémantique ayant servi de banc d'essai pour tester les propriétés applicatives issues de l'utilisation des technologies du Web sémantique. Cette application prend la forme d'un tableau de bord servant aux chercheurs en génie chimique de l'IREQ pour évaluer l'état de santé des transformateurs de puissance d'HQ grâce aux concentrations de certaines molécules présentes dans leur huile. Ce développement a permis de jeter les bases du cadre de développement d'applications auto-adaptatives.

Le CHAPITRE 4 présente un constructeur de requêtes visuelles conçu dans le but de démocratiser l'utilisation des technologies du WS chez HQ. Commandé par des chercheurs de l'unité de Science des données de l'IREQ, il devait permettre aux ingénieurs des unités d'affaires d'HQ de pouvoir explorer les bases de connaissances sémantiques sans en connaître le langage de requêtes. Plusieurs BDT ayant été constituées au fil de diverses recherches, un tel outil était souhaité afin de permettre à ces ingénieurs d'explorer ces données. Conçu pour ressembler aux constructeurs de requêtes visuelles du monde relationnel les plus connus, cet outil a permis l'étude des fonctionnalités souhaitables pour la création visuelle de requêtes.

Le CHAPITRE 5 présente un cadre de développement entrant dans la catégorie des constructeurs de vues et servant à générer des applications auto-adaptatives. Il a été utilisé pour générer une première application permettant le partage des données sur les matériaux autolubrifiants entre les ingénieurs qui conçoivent les centrales hydro-électriques, les chercheurs en sciences des matériaux et les techniciens qui font l'installation et la maintenance des pièces ainsi conçues. Développée dans le cadre du projet Tribologie, elle devait permettre aux chercheurs de l'IREQ d'enregistrer leurs résultats de recherche sur les matériaux autolubrifiants dans un outil en ligne. Ce dernier devait offrir la possibilité aux ingénieurs d'HQ de faire des recherches *ad hoc* sur ces résultats afin de choisir les meilleurs matériaux

pour des conditions environnementales données lors de la conception d'ouvrages hydro-électriques. Cette application a été mise en production dans les systèmes d'entreprise d'HQ.

Ces trois chapitres sont conçus de la même façon. Le contexte industriel est d'abord expliqué puis l'application développée est ensuite parcourue et documentée.

Le CHAPITRE 6 est une discussion comprenant des comparaisons des fonctionnalités des outils, expliquant les enjeux techniques de leur développement, exposant des réflexions théoriques issues de leur conception et présentant les leçons apprises et les limites de ces applications.

Finalement, la thèse se conclue par un retour sur les résultats, une présentation des retombées industrielles potentielles d'un tel cadre de développement, une discussion sur les limitations de la recherche et une présentation de travaux futurs qui pourraient découler de celle-ci.

0.2 Objectifs de la recherche

L'objectif principal de cette recherche était de concevoir une méthodologie permettant la génération d'applications s'auto-adaptant aux modèles de données en capitalisant sur les propriétés intrinsèques des bases de données sémantiques.

Cet objectif a été divisé en trois objectifs secondaires, chacun correspondant à une application logicielle développée dans le cadre d'un projet d'entreprise différent.

Le premier objectif secondaire était de fournir une preuve de concept et de faisabilité de l'objectif principal par le développement d'un navigateur sémantique présentant des fonctionnalités simples et des rudiments d'auto-adaptivité.

Le deuxième objectif secondaire était d'étudier les caractéristiques et les fonctionnalités d'une application générique très près du langage de requête de données. Il s'est matérialisé par le développement d'un constructeur de requêtes visuelles.

Le troisième objectif secondaire était de mettre à profit les connaissances acquises lors des deux premiers sous-objectifs afin de développer un cadre de développement pour applications auto-adaptatives offrant un équilibre entre utilisabilité et expressivité. Il a consisté à concevoir

un constructeur de vue issu de ce cadre de développement. Celui a été utilisé pour générer une première application.

0.3 Contributions

La principale contribution de cette recherche est le cadre de développement implémentant la méthodologie de génération d'applications auto-adaptatives. Ce cadre prend la forme d'un outil qui est désormais intégré aux systèmes d'entreprises d'HQ et pourra être réutilisé pour générer des applications.

À la connaissance de l'auteur, aucune solution actuelle ne permet de générer des applications de cette façon, autant au niveau de la couche serveur par les requêtes génériques, qu'au niveau de la couche client par l'utilisation de composantes auto-adaptatives et d'interfaces manœuvrables par des néophytes des technologies impliquées.

Contrairement à la majorité des solutions existantes pour la génération automatisée d'applications, dans le cas étudié aucun nouveau code ne sera généré. Comme il sera montré, d'autres méthodologies de génération d'applications existent, mais aucune ne génère de telles applications auto-adaptatives au modèle de données. C'est l'implémentation des propriétés d'auto-adaptativité au modèle de données qui permet à un canevas d'application d'être réutilisé pour créer de multiples applications, elles-mêmes auto-adaptatives.

Pour arriver à ce cadre, trois applications industrielles utilisant les technologies du Web sémantique et présentant différentes capacités d'auto-adaptativité ont été développées : DATE (Données d'analyse des transformateurs électriques), le CRV-HQ (Constructeur de requêtes visuelles d'HQ) et PPAL (Performance des paliers autolubrifiants) qui a été conçue grâce au cadre OM (Owl Monkey). Plusieurs contributions secondaires ont émané de ces implémentations. Le caractère appliqué de la recherche a contribué à la découverte et l'implémentation de plusieurs innovations afin de réussir à surmonter les problèmes rencontrés. Ces innovations sont de l'ordre des innovations de continuités et la plupart de celles-ci contribuent davantage à l'avancement de l'ingénierie qu'à l'avancement de la science

fondamentale. Ces contributions seront maintenant présentées, puis elles seront contextualisées plus en détails dans les prochains chapitres.

0.3.1 DATE

La principale contribution de cette application fut de démontrer qu'il est possible d'atteindre l'indépendance conceptuelle grâce aux technologies du WS. Le Tableau 1.1 présente comment les propriétés de ces technologies permettent d'atteindre les principes de l'indépendance conceptuelle. Il s'agit du cadre théorique justifiant ces recherches.

C'est à cette étape que l'utilité d'un véhicule générique servant à la communication entre le client et le serveur a été établie et que différentes pistes de conception ont été testées. L'hypothèse de l'utilité d'un tel véhicule pour atteindre l'indépendance conceptuelle s'est confirmée par l'utilisation de celui-ci sous différentes formes dans les trois applications de la recherche.

0.3.2 CRV-HQ

Les BDT étant différentes des BDR, il va de soi que les fonctionnalités qu'elles permettent soient différentes. La principale contribution émanant du développement de cette application est la transposition des fonctionnalités associées à la construction de requêtes SQL au langage SPARQL.

Une autre contribution provenant de la réalisation de cette application est la revue et l'étude des différentes fonctionnalités présentes dans des applications semblables, qu'elles soient relationnelles ou sémantiques. De plus, les fonctionnalités énumérées dans plusieurs cadres théoriques de visualisation ont aussi été étudiées. L'ensemble de ces fonctionnalités a été colligé dans plusieurs tables et forme un catalogue de fonctionnalités dont l'utilité et l'implémentation ont été discutées en fonction de l'expérience acquise. Enfin, de nouvelles fonctions rendues possibles par l'utilisation de BDT ont été recensées et étudiées, ou conçues.

La contribution la plus importante reste le constructeur de requêtes visuelles en tant que tel, qui a été conçu en utilisant les techniques de visualisation privilégiées par les systèmes relationnels de requêtes visuelles de façon à minimiser l'effort de transfert technologique des utilisateurs habitués aux outils relationnels standards. Cet effort qui semble aller de soi est pourtant absent de la littérature.

0.3.3 OM

C'est lors de cette étape qu'ont été mises ensemble toutes les connaissances accumulées lors des deux premières étapes afin d'instaurer la méthodologie de génération d'applications auto-adaptatives à l'intérieur d'un cadre de développement dirigé par les ontologies.

Le développement de ce cadre a permis d'expérimenter les implémentations d'une panoplie d'outils de visualisation et de manipulation de données sémantiques intuitifs, utilisables par des néophytes de ces technologies et utilisables dans des applications auto-adaptatives.

La première contribution issue de cet objectif est une composante logicielle de l'interface utilisateur qui permet la visualisation et l'interaction avec les données en graphe provenant d'une BDT : la grille sémantique. Cette nouvelle composante logicielle pour navigateur sémantique permet l'exploration ontologique par la construction itérative de l'ensemble des résultats. Ses diverses fonctionnalités n'ont sûrement pas atteint leurs formes finales, mais elles permettent tout de même d'appréhender la pertinence d'un tel outil. Elle permet en outre d'offrir une approche de découverte de connaissances générique pour les ontologies OWL.

La seconde contribution de cette partie de recherche est constituée des recommandations de modélisation ontologique permettant aux modèles d'être utilisés par des applications issues de cette méthodologie ou de méthodologies semblables.

La troisième contribution de ce chapitre est l'ontologie de visualisation (OV), qui est une première tentative vers la création d'un standard qui pourrait être réutilisé par la communauté du WS afin de guider et personnaliser les interfaces utilisateurs par des annotations faites à même les ontologies de domaine. Cela aurait le double avantage de faciliter la réutilisation des

composantes Web développées pour utiliser ces sémantiques et d'ainsi assurer une interopérabilité des ontologies par différentes plateformes applicatives.

Une quatrième contribution est constituée d'un jeu de pré-inférence représentant une solution simple et élégante au problème de la réconciliation du concept d'héritage dans les paradigmes orienté-objets avec le concept de classification du paradigme de la logique de premier ordre. Il sera présenté en détail au CHAPITRE 5.

Finalement, une autre contribution secondaire de cette étape se retrouve dans le fait d'offrir une approche permettant de croiser des données d'une ontologie OWL de manière *ad hoc*, ce qui, comme on le verra, pourra s'avérer une fonctionnalité de premier plan pour les entreprises 4.0 dans le cadre du Web sémantique des objets.

CHAPITRE 1

ÉTAT DES CONNAISSANCES

Afin de permettre la compréhension de cette recherche, la présentation des notions nécessaires a été divisée en trois parties : tout d'abord un survol des concepts théoriques est effectué, ensuite une revue de littérature est présentée et, finalement, le contexte industriel de la recherche est brièvement abordé.

1.1 Concepts théoriques

Cette recherche couvre plusieurs sous-domaines de l'informatique et, dans une approche de haut en bas, seront tout d'abord présentées certaines définitions de la science de l'information, pour ensuite parcourir les concepts plus spécialisés des systèmes d'information puis des technologies du Web sémantique.

1.1.1 Science de l'information

Saracevic propose comme définition de la science de l'information qu'elle est "la science et la pratique étudiant la collecte, l'entreposage, la récupération et l'utilisation de l'information et qu'elle s'intéresse à l'information et aux connaissances enregistrables et aux technologies et services qui facilitent leur gestion et leur utilisation" (Saracevic, 2009). Bawden et Robinson (Bawden & Robinson, 2015) décrivent la science de l'information en tant que champ d'étude, au sens de Hirst (Hirst, 1974), c'est-à-dire qu'elle se concentre sur un sujet d'intérêt en utilisant n'importe quelles formes de savoir (sociologique, mathématique, philosophique, etc.) qui lui est pertinent, par opposition aux disciplines qui se basent sur une seule forme de savoir.

Bawden et Robinson disent de la science de l'information qu'elle est "un champ d'étude multidisciplinaire, impliquant plusieurs formes de savoir, dont la cohérence est donnée par leur concentration sur le concept central qu'est l'information humaine enregistrée" (Bawden & Robinson, 2015). Dans le même ouvrage, les auteurs citent les nombreuses vues sur ce qu'est

cette science multidisciplinaire dans la littérature : “une méta-science, une inter-science, une science post-moderne, une science interface, une science supérieure, une science rhétorique, une science nomade, un sujet interdisciplinaire qui devrait être renommé science du savoir, un sujet qui pourrait assumer le rôle qui a été joué par la philosophie dans les sciences des médias et de l’humanisme” (Bawden & Robinson, 2015).

Floridi (Floridi, 2002) parle quant à lui de philosophie appliquée de l’information. Les sujets étudiés par cette science varient aussi selon la littérature et peuvent comprendre : l'interaction humain-machine, la littératie informationnelle, la gestion de l’information, la documentation, la gestion bibliographique, la gestion des connaissances, l’organisation de l’information, l’étude sociétale de l’information, la bibliométrie, la recherche d’information et la récupération de données (Bawden & Robinson, 2015).

La nature pluridisciplinaire de la science de l’information, et les nombreuses sciences qu’elle chevauche et auxquelles elle emprunte du savoir et des techniques permet d’appréhender la nature multidisciplinaire des systèmes d’information eux-mêmes.

1.1.2 Systèmes d’information

Tout comme la science de l’information, plusieurs définitions sont disponibles dans la littérature sur ce qu’est un système d’information. Zwass décrit un système d’information comme étant “un ensemble intégré de composantes pour colliger, entreposer et traiter des données et pour fournir de l’information, du savoir et des produits numériques (Zwass, 2020).” Ces composantes sont selon lui : le matériel informatique (*hardware*), les logiciels (*software*), les télécommunications, les bases et entrepôts de données ainsi que les ressources humaines et les procédures. Le *BusinessDictionnary* (Information system, 2020), quant à lui, donne la définition suivante : “une combinaison de matériel informatique, de logiciels, d’infrastructures et de personnels formés, organisés pour faciliter la planification, le contrôle, la coordination et la prise de décision dans une organisation.” Le *Web Dictionary of Cybernetics and Systems* (Heylighen, 2020) propose une définition plus vaste : “Un système de fonctions concernant l’acquisition et le transfert de l’information dont le transporteur peut être une unité biologique, personnelle, sociale ou technique. Un système d’information est dédié à une certaine forme

d'information (un sujet), même si celui-ci peut être très large. Dans la plupart des cas, un outil d'entreposage fait partie d'un système d'information."

Cette dernière définition est intéressante puisqu'elle introduit la notion de spécialisation d'un système d'information à une certaine forme d'information. Dans le contexte de la présente recherche, une méthodologie sera présentée pour développer des applications à l'aide de composantes génériques pouvant s'adapter à n'importe quel domaine de savoir. Toutefois, les applications créées s'appuieront sur des modèles de données particuliers représentant effectivement un champ précis du savoir, une forme précise d'information.

Dans le contexte de la présente recherche, ce sont les composantes logicielles et les bases de données des systèmes d'information qui seront étudiées. L'ensemble de ces composantes forment des applications d'architectures trois-tiers (Architecture trois tiers, 2020) dont les fonctionnalités particulières reposent sur les technologies du WS et plus particulièrement les BDT. En effet, ces technologies ont des particularités intrinsèques qui permettent des fonctionnalités d'auto-adaptabilité.

1.1.3 Technologies du Web sémantique

L'idée du WS a été formalisée par Tim Berners-Lee, l'inventeur du *World Wide Web*, dans un article paru en 2001 (Berners-Lee, Hendler, & Lassila, 2001). Le Web sémantique y est décrit comme l'évolution d'un Web de documents vers un Web de données. Fondé par M. Berners-Lee lui-même, le *World Wide Web Consortium* (W3C) (W3C, 2020) est une communauté internationale dont la mission est d'amener le Web à réaliser son plein potentiel. Leur site décrit le WS comme un cadre de développement commun permettant aux données d'être partagées et réutilisées au-delà des frontières des applications, des entreprises ou des communautés (Semantic Web, 2020). Le WS est basé sur l'utilisation du langage *Resource Description Framework* (RDF) : un modèle standard d'échange de données ayant des caractéristiques facilitant leur intégration et supportant spécifiquement l'évolution des modèles dans le temps (RDF, 2020). Ce standard utilise des identifiants de ressource uniformes (en anglais *Uniform Resource Identifier*, URI) qui sont des chaînes de caractères identifiant une ressource sur le Web (*e.g.* : http://www.etsmtl.ca/).

Tel que mentionné, le WS a pour objectif de construire une pile technologique permettant de passer d'un Web de documents à un Web de données. Cette pile technologique contient, entre autres, les technologies RDF (RDF, 2020), *SPARQL Query Language for RDF* (acronyme récursif : SPARQL) (SPARQL, 2020), *Resource Description Framework Schema* (RDFS) (RDFS, 2020), et *Web Ontology Language* (OWL) (OWL, 2020).

Les données RDF peuvent être stockées dans des BDT. Le triplet est l'unité fondamentale du langage RDF. Il s'agit d'une déclaration composée de trois parties : le sujet, le prédicat et l'objet. Le sujet et le prédicat sont des URI représentant respectivement la ressource dont il est question et la relation qu'il a avec l'objet. L'objet peut être un URI ou un littéral et peut donc représenter, respectivement, une ressource ou une valeur.

Malgré l'utilisation d'URI, les triplets ne sont pas nécessairement déréférençables, c'est-à-dire qu'ils peuvent ne pas être des liens internet valides. Une collection de triplets forme un graphe orienté dont les arcs sont étiquetés et représentent les relations entre les nœuds que sont les ressources.

RDFS est un langage de représentation des connaissances permettant de structurer les ressources RDF dans des ontologies. La définition la plus largement répandue d'une ontologie est celle de Gruber (Gruber, 1995) : « Une ontologie est la spécification d'une conceptualisation. [...] Une conceptualisation est une vue abstraite et simplifiée du monde que l'on veut représenter. » RDFS permet d'agencer les sémantiques de RDF pour former un métalangage de représentation de la réalité. Y sont introduits, entre autres, les concepts de relations entre les classes, qu'elles soient de subsomption ou d'association. Alors que les relations de subsomption permettent d'indiquer quelles classes sont des sous-classes ou superclasses d'autres classes, les relations d'association permettent de relier des classes issues ou non de différentes hiérarchies existentielles entre elles.

OWL est un langage qui s'utilise par-dessus RDFS pour représenter des connaissances plus complexes sur des objets, des groupes d'objets ou des relations entre des objets. L'expressivité d'OWL permet d'utiliser le plein potentiel de la logique de prédicat dans la construction des ontologies. RDFS et OWL sont tous deux fondés sur la logique de premier ordre et permettent

l'utilisation d'engins d'inférence afin d'expliciter des informations implicites contenues dans les données en plus de valider leur cohérence logique.

OWL est si expressif qu'il compromet la décidabilité des engins d'inférence. C'est pourquoi plusieurs saveurs d'OWL ont été élaborées, celles-ci sacrifiant l'expressivité du langage au profit de l'efficience computationnelle et de la résolubilité. Alors qu'OWL 2 / FULL est la version complète du langage pour lequel aucun algorithme décidable existe, trois sous-ensembles de celui-ci (OWL 2 EL, OWL 2 QL et OWL 2 RL) sont quant à eux décidables et utilisés dans différents cas d'application. OWL 2 EL sera privilégié lorsqu'une ontologie présente beaucoup de classes et de propriétés liées de façons complexes entre elles. OWL 2 EL permet alors de déterminer en temps polynomial les relations entre les classes et l'appartenance des individus aux classes. On utilisera OWL 2 QL lorsqu'une ontologie présente un nombre élevé d'individus et OWL 2 RL pour travailler avec des engins à base de règles traditionnelles. L'engin d'inférence d'Oracle utilisé dans la présente recherche offre sa propre saveur d'OWL, appelée OWLPRIME, qui présente un sous-ensemble des règles OWL pouvant être utilisées efficacement avec ses technologies.

On peut utiliser RDFS et OWL pour construire des ontologies. Une ontologie est dans ce cas définie comme une collection de triplets formant un graphe orienté qui représente les connaissances d'un domaine spécifique. Elle peut n'être qu'une coquille vide représentant la hiérarchie et la structuration des connaissances de ce domaine, mais elle peut aussi contenir les individus des classes de ce domaine, agissant donc à titre de base de données pouvant être interrogée.

Dans OWL, trois types de propriétés permettent de construire la charpente de l'ontologie : les propriétés de type données, les propriétés de type objets et les propriétés de subsomption. Les propriétés de type données sont représentées par la sémantique *owl:DatatypeProperty* et elles servent à lier des individus à des valeurs littérales. Les propriétés de type objets sont représentées par la sémantique *owl:ObjectProperty* et permettent de lier les individus entre eux. Finalement, les propriétés de subsomption permettent de lier les classes à leurs sous-classes et superclasses grâce à la sémantique *rdfs:subClassOf*.

On retrouve aussi deux autres types de propriétés : les propriétés d'annotations et les propriétés ontologiques. Les premières servent à donner des informations supplémentaires sur les objets ontologiques, telles que des commentaires ou étiquettes. Elles ne sont pas considérées par les engins d'inférence. Les secondes sont des propriétés d'information sur l'ontologie; elles permettent d'indiquer notamment les versions de celle-ci, les sémantiques obsolètes ou encore à pointer vers d'autres ontologies dont les sémantiques sont utilisées.

La définition des propriétés contient toujours un domaine et un codomaine, peu importe leur type. Le domaine des propriétés de type données et de type objets est toujours une classe. Les individus de cette classe pourront donc être caractérisés par cette propriété.

Le codomaine d'une propriété de type donnée est un type de données littérales, comme une chaîne de caractères, un nombre entier, etc. Ces types de données littérales peuvent être fabriqués par l'ontologiste mais sont généralement issus de l'ontologie *XML Schema Definition* (XSD), un autre standard du W3C spécifiant comme décrire formellement des éléments du langage *Extensible Markup Language* (XML). On pourra vérifier la validité des données de l'ontologie entre autres grâce à ces types.

Le codomaine d'une propriété de type objet est une classe. Les individus d'une classe pourront être liés aux individus d'une autre classe par une propriété objet ayant respectivement comme domaine et comme codomaine ces deux classes.

Afin d'assurer la validité d'une ontologie OWL, on déclare une classe à la racine de toutes les classes : *owl:Thing*. Il s'agit de la superclasse de toutes les classes. De même, afin d'assurer la fermeture des raisonnements logiques, on utilise la classe *owl:Nothing* comme étant la fin de toutes les arborescences de subsomption.

SPARQL est le langage de requêtes permettant les opérations de création, de lecture, de modification et de suppression de données (en anglais, *Create, Read, Update, Delete*, CRUD) sur les collections de RDF. Il est l'équivalent du SQL (en anglais, *Structured Query Language*) pour les BDT et il lui ressemble d'ailleurs beaucoup.

SPARQL utilise principalement trois types de requêtes : SELECT, CONSTRUCT et ASK. Le type SELECT, comme en SQL, permet de récupérer les données de l'ontologie sous forme de table en deux dimensions. Les requêtes CONSTRUCT quant à elles, ramènent l'information

sous forme de triplets et conserve donc le format de graphe dans les résultats. Finalement, les requêtes ASK retourne un booléen comme résultat.

Puisque les ontologies ont été conçues avec en tête l'idée d'être utilisées pour le Web, des librairies permettent de sérialiser les résultats des requêtes directement sous format *JavaScript Object Notation* (JSON), largement utilisé dans les applications Web. Alors que les résultats des requêtes SELECT sont directement exprimés en format JSON, les résultats des requêtes CONSTRUCT peuvent être exprimés dans le format JSON-LD (*JSON for Linked Data*) qui préserve la structure en graphe des données. Une librairie telle que JSON-LD.js 1.0 (JSON-LD, 2020) permettra de faire le passage entre les formats JSON et JSON-LD.

SPARQL utilise plusieurs mots-clés semblables à SQL tels que DISTINCT, GROUP BY, etc. Cependant, d'autres sémantiques lui sont particulières telles qu'OPTIONAL qui permet en quelque sorte de faire des jointures externes ou SERVICE qui permet d'interroger plusieurs points de terminaison SPARQL. Ce qui distingue le plus SPARQL de SQL est assurément la forme du corps des requêtes qui se construit dans le premier cas par l'accumulation de patrons RDF. Un patron est composé de trois parties comme un triplet : sujet, prédicat, objet, dont zéro, plusieurs, ou toutes ces parties sont remplacées par des variables. L'assemblage de multiples patrons permet de représenter des morceaux du graphe de l'ontologie et donc de regrouper tous les triplets correspondant à la composition de ces patrons.

Afin de faciliter la réutilisation des ontologies, il est de pratique usuelle de persister les données dans un graphe distinct de celui contenant l'ontologie. On pourra ainsi, entre autres, joindre une même ontologie avec des jeux de données différents. Par l'utilisation des sémantiques de l'ontologie dans les données (noms des classes, noms des propriétés, etc.) et étant donné l'utilisation de triplets, le simple fait de mettre dans le même contenant virtuel l'ontologie et les données donne un graphe unique dans lequel ces deux graphes sont imbriqués.

Dans la communauté du WS, le graphe du modèle de données est appelé TBox (pour *Terminology Box*) et le graphe des données est appelé ABox (pour *Assertions Box*). On retrouve dans la littérature plusieurs autres types de *Box*, mais dans le cadre de cette recherche, deux autres types ont été définis et utilisés : le SBox (*Save Box*) et le EBox (*Enumerations Box*). Leurs utilités seront discutées ultérieurement.

L'utilisation d'ontologies est de plus en plus fréquente, en raison de ses propriétés qui, à terme, devraient permettre l'avènement du Web sémantique, selon la vision de Tim Berners-Lee et du W3C. Ces propriétés sont : la facilité de joindre ensemble deux collections de RDF, la possibilité de faire de l'inférence, l'indépendance des concepts avec les langues, les possibilités de réutilisation des ontologies et le fait que les machines peuvent se passer des humains lors de l'échange d'information grâce à la description formelle de chacun des éléments par les URI.

Plusieurs œuvres sont de bons points de départ pour se familiariser avec ces technologies dont Antoniou, Groth, Harmelen, & Hoekstra (2012), Allemang & Hendler (2008) et Paquette (2010). Ces œuvres couvrent aussi les capacités d'inférence qui proviennent de l'utilisation de RDFS et OWL. Un des concepts jouant un rôle fondamental dans l'inférence est l'hypothèse du monde ouvert.

Cette hypothèse utilisée en logique formelle est utilisée pour signifier que la véracité d'un fait ne dépend pas de la connaissance actuelle qu'on a d'un système. Elle peut se traduire par l'adage : l'absence de preuve n'est pas une preuve d'absence. Elle s'oppose à l'hypothèse du monde clos dans laquelle on peut déduire qu'une affirmation est fausse si on n'a pas d'information sur celle-ci. Les BDR fonctionnent généralement dans un monde clos. Posons par exemple qu'une base de données ne contient qu'une seule information à savoir que Luc vit à Ste-Julie. Dans un monde clos, en formulant une requête pour savoir si Christian vit à Ste-Julie, la réponse sera négative. Cependant, dans un monde ouvert, cette question n'aura tout simplement pas de réponse.

Le concept de WS a permis l'émergence des données ouvertes et liées (DOL, en anglais *Linked open data (LOD)*) (Linked data, 2020). Ce projet du W3C s'inscrit dans le mouvement des données ouvertes (Données ouvertes, 2020) et a comme but d'étendre le Web avec des données communes publiées de plusieurs sources sous le format RDF et liées entre elles par des triplets unissant les différents jeux de données. Ces liens permettent des analyses communes sur les ensembles de données et sont des chemins que peuvent suivre les robots d'indexation (*Web crawlers*). En 2016, Ermilov *et al.* recensaient environ 150 milliards de triplets provenant de près de 3000 jeux de données ouvertes et liées (Ermilov, Lehmann, Martin, & Auer, 2016). Des diagrammes des nuages formés par ces jeux de données montrant l'évolution de ce

mouvement entre les années 2007 et 2020 sont disponibles en ligne (The Linked Open Data Cloud, 2020). Au centre de ces diagrammes on retrouve DBPedia (Learn about DBPedia, 2020), un projet citoyen (*crowd-sourced*) pour extraire des données structurées de Wikipédia (Wikipedia, 2020). La version anglophone de DBpedia décrit 4,58 millions de choses dont 4,22 millions sont structurées dans une ontologie commune.

1.2 Revue de littérature

Le présent travail réunit des notions provenant de plusieurs domaines de recherches. De plus, y sont implémentés des outils informatiques de différents types. Maintenant seront présentés les ouvrages phares couvrant ces domaines et ces types d'outils.

1.2.1 Propriétés auto-*

Puisque des applications Web puisant leurs données dans des BDT peuvent être construites, il semble alors naturel de tenter d'utiliser les propriétés de ces bases de données pour augmenter le potentiel de ces applications. Les propriétés d'auto-adaptativité dont il sera question dans cette recherche entrent dans la catégorie des propriétés permettant à un système de se maintenir, appelées en anglais les *self-* properties*, qui seront dans la suite de ce texte nommées les propriétés auto-*.

Cette section traite de ces propriétés auto-*. Elle est en partie une traduction de "*Toward an Adaptive Application Builder: Two Communication Systems for an Ontology-Based Adaptive Information System Framework*" (Bhérer, Vouligny, Gaha, & Desrosiers, 2016), à la section "*II. Related Word*", sous-section "*B. Self-* properties*".

En 2001, IBM a proposé l'Initiative d'informatique autonome (*Autonomic Computing Initiative*) (Horn, 2001) avec l'objectif de développer des mécanismes pour permettre aux systèmes et sous-systèmes de s'auto-adapter à des changements imprévisibles. Des conférences telles que *Software Engineering for Adaptive and Self-Managing Systems* (SEAMS, 2020) montrent que l'auto-adaptativité des systèmes et des logiciels est encore un

domaine de recherche important, aujourd'hui divisé en une variété de sous-domaines. Parmi ceux-ci, on pourrait inclure les capacités d'adaptation d'une application à l'évolution de son modèle de données.

Comme Dobson *et al.* (2010) l'ont pointé, l'Initiative d'informatique autonome n'a pas tenu les promesses annoncées (Kephart & Chess, 2003). Même si plusieurs avancées individuelles ont amené quelques-uns des bénéfices attendus, il n'y a toujours pas de solution intégrée menant à un système autonome. C'est une tâche que certains chercheurs ont entreprise, comme Bermejo-Alonso, Sanz et Hernádez (2010) avec leur ontologie pour l'ingénierie des systèmes autonomes. Toutes ces avancées individuelles tombent dans la catégorie des propriétés auto-*, i.e., des façons pour les systèmes de se maintenir automatiquement à travers différents scénarios (Berns & Ghosh, 2009).

Les mécanismes d'autoadaptation que la présente recherche propose pourraient aider au développement de propriétés d'auto-conscience (*self-aware properties*) ou d'auto-ajustement (*self-adjusting properties*) (Sterritt & Hinchey, 2010), menant au développement de composantes autonomes. À l'intérieur des hiérarchies des propriétés auto-* de Salehie et Tahvildari (2009) et de Berns et Ghosh (2009), ces mécanismes se classifieraient comme un ensemble de propriétés d'auto-configuration (*self-configuring properties*). Une fois ces mécanismes intégrés dans un outil de construction d'applications, des propriétés d'auto-optimisation (*self-optimizing properties*) sont susceptibles d'émerger.

1.2.2 Indépendance conceptuelle

McGinnes et Kapros (2015) circonscrivent le problème de la non-adaptabilité des applications comme une dépendance conceptuelle avec le modèle de données. Ils décrivent cette dépendance entre le modèle de données et l'application comme un couplage logiciel indésirable. Les auteurs utilisent le terme "système d'information adaptatif"`(SIA) (*adaptive information system*) pour désigner un système d'information qui s'adapte aux changements fait dans son modèle de données sous-jacent. Ils concluent que la plupart des applications basées sur des systèmes d'information utilisées aujourd'hui sont dépendantes du modèle de leur domaine. De tels systèmes doivent donc être maintenus à chaque fois que le modèle de

données change et même le plus petit changement peut potentiellement entraîner une adaptation manuelle longue et coûteuse.

McGinnes et Kapros (2015) proposent six principes pour atteindre l'indépendance conceptuelle sur n'importe quelle source de données. En utilisant ces principes, ils montrent qu'il est possible de bâtir un SIA basé sur le langage XML cartographiant une BDR. Ces principes ont été appliqués aux technologies du WS (voir Tableau 1.1) dans Bhérer, Vouligny, Gaha, Redouane, & Desrosiers (2015), ce qui semble constituer une base argumentaire solide pour défendre l'utilisation de ces technologies dans le développement des SIA. En effet, atteindre l'indépendance conceptuelle en utilisant des technologies basées sur le RDF telles que RDFS ou OWL semble plus simple et intuitive qu'en utilisant des sources de données relationnelles. Ce qui est certain, c'est que les technologies basées sur le RDF possèdent inhéremment plusieurs des propriétés requises intégrées à leur conception, réduisant ainsi la complexité pour atteindre l'indépendance conceptuelle.

L'atteinte de cette indépendance conceptuelle a un double avantage. Le premier est d'obtenir des applications auto-adaptatives alors que le second est que les tiers client et serveur de telles applications sont en fait une collection de composantes génériques pouvant s'utiliser avec virtuellement n'importe quelle ontologie, peu importe le domaine qu'elle décrit (Bhérer, Vouligny, Gaha, Redouane, & Desrosiers, 2015). Il devient alors possible de générer des applications par l'assemblage de ces composantes avec des ontologies de domaine.

Tel qu'annoncé, la présente recherche tentera de formaliser une méthodologie pour automatiser la création d'applications auto-adaptatives. Le rêve d'une production complètement automatisée d'applications à partir des modèles est le but ultime de l'ingénierie dirigée par les modèles (IDM). L'IDM souhaite que les changements apportés au modèle entraînent la modification automatique du code des applications utilisant ce modèle. La méthodologie proposée ici tentera de résoudre ce problème non pas en modifiant le code des applications mais en utilisant des fonctions auto-adaptatives déjà codées pour s'adapter à ces changements. L'ingénierie dirigée par les ontologies (IDO) est, quant à elle, le pendant sémantique de l'IDM. La prochaine section traitera donc de l'IDM et de l'IDO. Elle est en partie une traduction de

Bhérer, Vouligny, Gaha, & Desrosiers (2016) à la section "*II. Related Word*", sous-section "*A. Model-driven engineering and ontology-driven architecture*".

Tableau 1.1 Principes de l'indépendance conceptuelle
Source : Adapté de Bhérer *et al.* (2016)

PRINCIPES DE L'INDÉPENDANCE CONCEPTUELLE	**APPLICATIONS DES PRINCIPES AUX TECHNOLOGIES BASÉES SUR LES RDF**
1. Fonctionnalités réutilisables (comportement structurellement approprié): le SIA peut prendre en charge tout modèle conceptuel. Le code et les structures dépendant du domaine sont évités. Une fonctionnalité générique est appelée à l'exécution pour chaque type d'entité.	Ce principe s'applique de la même manière en utilisant une BDT. Les requêtes SPARQL génériques sont obtenues en codant en dur exclusivement les sémantiques RDF, RDFS ou OWL. Le modèle de données peut être inspecté dynamiquement lors de l'exécution à l'aide de requêtes SPARQL génériques.
2. Catégories de données connues (comportement sémantiquement approprié): chaque type d'entité est associé avec une ou plusieurs catégories génériques prédéfinies. La fonctionnalité spécifique à la catégorie est appelée lors de l'exécution pour chaque type d'entité.	Toutes les ontologies utilisant les langages RDFS ou OWL contiennent *ipso facto* la même base conceptuelle. Ces méta-entités sont définies par RDF, RDFS et OWL. En utilisant ces méta-entités comme les entités les plus génériques du SIA, toute ontologie basée sur les RDF peut être utilisée. McGinnes et Kapros utilisent des catégories archétypales tirées du domaine de la psychologie pour classer les entités selon les comportements que le SIA devrait adopter en leur présence. Cette idée intéressante n'a pas été examinée ici.
3. Gestion adaptative des données (évolution de schéma): Le SIA peut stocker et réconcilier des données avec plusieurs définitions pour chaque type d'entité (c.-à-d. plusieurs modèles conceptuels), permettant à l'utilisateur final de comprendre le sens des données.	Premièrement, la technologie RDF utilise ce que McGinnes et Kapros appellent des «schémas mous», les modèles de données étant stockés sous la forme de données. Deuxièmement, la technologie RDF permet aux individus avec différentes propriétés de coexister dans la même classe. De plus, les individus peuvent appartenir à plus d'une classe. Les axiomes comme *owl:sameAs* ou *owl:equivalentClass* rendent possible la réconciliation des données d'entités décrites distinctement. Deux classes décrites de façons différentes déclarées équivalentes auront, par inférence, le même ensemble de propriétés et alors deux individus de cette nouvelle classe pourront n'avoir que des propriétés évaluées différentes. Ce mécanisme prend ainsi en charge la réconciliation des données de différents modèles conceptuels. Alors que le modèle évolue, les données utilisant différents modèles conceptuels restent disponibles et sont instantanément accessibles sans aucune refactorisation du SIA.
4. Imposition du schéma (intégrité de domaine et référentielle): chaque donnée stockée se conforme à une définition de type d'entité particulière, qui lui a été attribuée au moment de la saisie des données (ou de la dernière modification).	Dans les technologies telles que OWL, l'intégrité du domaine et l'intégrité référentielle peuvent être validées avec des raisonneurs. En ce qui concerne les types de données, les données littérales sont généralement associées à des types de base lors de l'entrée dans une BDT.
5. Identification de l'entité (intégrité de l'entité): les données relatives à chaque entité sont identifiées de manière unique et invariante par rapport aux changements du modèle de données.	Dans la technologie RDF, l'identification d'entité est assurée par le mécanisme des URI, et elle se fait donc déjà de manière unique et invariante par rapport aux changements du modèle.
6. Étiquetage (gestion des données): les données stockées relatives à chaque entité sont étiquetées de telle sorte que les modèles conceptuels applicables peuvent être déterminés.	En utilisant la technologie RDF, ce principe signifie que chaque individu doit appartenir à une classe. Dans ce cas, peu importe de quelle façon la classe change au fil du temps, tous ses individus peuvent avoir n'importe quel nombre de propriétés évaluées ou non. Cependant, des étiquettes lisibles par les humains sont nécessaires pour présenter l'information aux utilisateurs et il est obligatoire d'attribuer ces étiquettes à chaque entité.

1.2.3 Ingénierie dirigée par les modèles et ingénierie dirigée par les ontologies

Depuis les années 1990, les méthodologies IDM ont tenté de permettre le développement logiciel à partir des modèles. De façon à maximiser la productivité, le modèle

de l'application logicielle serait idéalement suffisant à certains programmes pour qu'ils génèrent automatiquement le logiciel lui-même. Même si l'IDM a eu quelques succès, elle n'est pas devenue une approche universelle, surtout parce que son adoption est longue et complexe. Malgré cela, plusieurs chercheurs et compagnies continuent d'investir des efforts dans l'IDM dû aux grands bénéfices qu'elle laisse présager (Gherbi, Meslati, & Borne, 2009). Douglas C. Schmidt (2006) décrit l'IDM comme une approche prometteuse pour protéger les développeurs de la complexité des plateformes, de la même manière que les premiers langages de programmation ont protégé les programmeurs de la complexité du code machine. Schmidt affirme que les langages de troisième génération ont échoué à soulager de cette complexité en raison de leur propre complexité et de l'évolution rapide et la prolifération des plateformes.

Les ontologies ont, quant à elles, été un sujet de recherche populaire ces dernières années, surtout en raison de leurs capacités d'interopérabilité qui pourrait faciliter l'avènement du WS. Comme une ontologie est la "description des concepts et des relations qui peuvent exister formellement pour un agent ou une communauté d'agents" (Gruber, 1995), il est aisément compréhensible que plusieurs chercheurs aient tenté de les utiliser avec l'IDM.

L'*Object Management Group* (OMG) (OMG, 2020), promoteurs de l'architecture dirigée par les modèles (ADM), déclarent que cette approche vise la réutilisation des applications, la réduction de la complexité, l'interopérabilité entre les plateformes, la spécificité des domaines et l'indépendance des plateformes (MDA, 2020). Comme de tels objectifs sont aussi des objectifs des technologies sémantiques, on peut présumer l'émergence d'une synergie en combinant ces deux paradigmes. Le W3C, fournisseur des fondations du Web sémantique, suggère que même si l'ADM est un bon cadre de développement pour le développement logiciel, il pourrait être amélioré par l'utilisation des technologies sémantiques pour désambiguïser le vocabulaire des domaines, valider la consistance des modèles et augmenter l'expressivité de la représentation des contraintes. En augmentant ainsi la pile méthodologique de l'OMG, les ontologies pourraient favoriser l'émergence de l'IDO.

Pan, Staab, Aßmann, & Ebert (2013) proposent d'utiliser les ontologies et l'ADM ensemble pour obtenir le meilleur des deux mondes. Leur approche consiste à construire des ponts entre les espaces techniques des "outils modèles" et des "outils ontologiques". Ces deux espaces

sont construits sur différentes couches : du métalangage (M3) aux langages (M2), aux modèles (M1), jusqu'aux instances courantes (*running instances*) (M0). Par la construction de ponts entre les couches, les capacités des ontologies peuvent ainsi être utilisées dans une approche de développement logicielle ADM. Les ontologies sont alors intégrées dans le développement logiciel dirigé par les modèles pour valider la consistance des implantations, guider le développement logiciel et relier causalement les spécifications durant le processus de développement (Pan, Staab, Aßmann, & Ebert, 2013).

Dans ce qui semble une approche plus simple pour relier les deux mondes, Martins Zviedris *et al.* décrivent comment ils ont pu construire automatiquement des applications basées sur des ontologies (Zviedris, Romane, Barzdins, & Cerans, 2014). En se basant sur le principe de Sowa (Sowa, 2011) qui dit que tout logiciel comporte une ontologie, implicite ou explicite, les auteurs croient qu'il est possible de développer un méta-modèle universel et indépendant des plateformes en utilisant une approche ADM.

Ils procèdent en développant tout d'abord une ontologie des applications Web qu'ils instancient chaque fois qu'ils veulent construire une nouvelle application. Un engin qu'ils ont construit est ensuite utilisé pour "comprendre" l'instance d'une application en particulier et générer automatiquement son code. Le résultat est une application codée en dur non adaptative avec une cartographie un pour un des classes et propriétés de l'ontologie de domaines en classes et attributs JAVA (Java, 2020). L'application peut être reconstruite si un changement est apporté au modèle de données mais celle-ci doit alors être recompilée avant utilisation.

La principale différence entre les travaux de Pan *et al.* (2013) et de Zviedris *et al.* (2014) est que les premiers tentent d'amener les capacités des ontologies dans le monde de l'ADM alors que les seconds se focalisent sur l'implémentation des apprentissages apportés par l'ADM dans les standards sémantiques. Même si l'approche de Pan *et al.* d'établir des ponts entre les standards permet l'utilisation des nombreux outils existant de l'ADM avec les deux technologies, elle est de loin plus complexe à implémenter. En ignorant les standards de l'OMG, Zviedris *et al.* amènent plus facilement les avantages d'une approche ADM dans le royaume sémantique, au coût de devoir recoder des outils déjà existants dans la trousse de

l'ADM, i.e., leur constructeur d'application et leur engin d'exécution de ces applications. Ces deux équipes approchent donc l'IDO par des directions opposées.

La méthodologie qui sera proposée par cette recherche peut être aussi considérée comme une approche IDO différente des deux présentées précédemment. Elle est du même côté du spectre que l'approche de Zviedris *et al.* puisqu'elle tente d'incorporer les découvertes de l'ADM dans un monde sémantique. Elle en diffère cependant radicalement. Alors que l'approche de Zviedris *et al.* tente de faire correspondre le contenu ontologique à des éléments du paradigme de programmation orienté-objet (OO) de façon automatisée, la méthodologie proposée impliquera l'utilisation de code générique s'adaptant au contenu ontologique et permettant de parcourir celui-ci sans que du nouveau code soit généré. Parmi ces composantes génériques, on retrouve inévitablement celles permettant la visualisation des données sémantiques par les utilisateurs.

1.2.4 Paradigmes de visualisation de l'information

Alors que le WS commence à prendre de l'ampleur, plusieurs éléments sont encore souhaitables pour faciliter son adoption. Peinl dans son récent article sur l'état de l'art du WS et son adoption dans les entreprises souligne que des interfaces graphiques pour utilisateurs intuitives pour les applications sémantiques sont encore au stade embryonnaire et qu'idéalement celles-ci devraient prendre avantage des technologies modernes du Web (Peinl, 2016). Dadzie et Pietriga (2017) arrive à un constat similaire indiquant qu'il faut que se développent de nouvelles façons d'interagir avec le très large volume de données complexes, liées et multidimensionnelles tout au long de leur cycle de gestion. Plusieurs articles scientifiques recensent des paradigmes de visualisation de données sémantiques (Katifori, Halatsis, Lepouras, Vassilakis, & Giannopoulou, 2007; Mangold, 2007; Dadzie & Rowe, 2011; Hachey & Gasevic, 2011; Marie & Gandon, 2014; Grafkin, *et al.*, 2016; Vega-Gorgojo, *et al.*, 2016). Avant de se lancer dans l'étude de ces paradigmes, il convient de commencer par explorer différentes taxonomies des techniques de visualisation en général.

Plusieurs classifications des techniques de visualisation existent dans la littérature (Shneiderman, 1996; Tory & Moller, 2004; Liu, Cui, Wu, & Liu, 2014; Guimaraes, Freitas, Sadre, Tarouco, & Granville, 2015). Guimaraes, Freitas, Sadre, Tarouco, & Granville (2015) expliquent que l'étude des formes que prend la visualisation de l'information est un champ de recherche en soi dans les sciences informatiques. Cette discipline a été reconnue formellement à la fin des années 1980. Card (1999) la définie comme "l'utilisation de représentations visuelles de données abstraites sur un support informatique interactif pour amplifier la cognition" alors que Ward *et al.* (2010) la définissent comme "le processus de représenter des données, de l'information ou des connaissances de façon visuelle pour supporter les tâches d'exploration, de confirmation, de présentation et de compréhension".

Guimaraes, Freitas, Sadre, Tarouco, & Granville (2015) spécifient qu'historiquement ce champ de recherche se divise en deux sous-champs principaux : la visualisation scientifique qui traite des données scientifiques souvent associées à une composante spatiale et la visualisation de l'information qui traite des données abstraites normalement sans composante spatiale. Les techniques de visualisation scientifique étaient alors catégorisées par le nombre de variables indépendantes présentes dans les données et le type de ces données (scalaire, vecteur, tenseur ou multivariée). Les techniques de visualisation de l'information, quant à elles, étaient catégorisées uniquement selon leur type dont des catégories typiques sont les bases des données multidimensionnelles (1, 2, 3…), les textes, les graphes et les arbres.

Tory et Möller (2004) avancent que la distinction, usuelle à l'époque, entre la visualisation scientifique et la visualisation de l'information porte à confusion puisqu'il y a des recoupements entre ces deux champs, que les termes sont mal choisis, contredisent l'intuition et qu'elle force une division du domaine qui peut décourager les recherches dont l'objet se trouve à la fois dans les deux sous-champs. Ils présentent donc une taxonomie de haut niveau basée sur les conceptions de modèles de données au lieu de types de données. Ces conceptions sont d'abord séparées en deux catégories selon que l'objet de l'étude soit discret ou continu. Cet aspect est important selon eux puisque les modèles continus assument que les données peuvent être interpolées alors que les modèles discrets ne le permettent pas. Des cas comme les ratios et les intervalles se retrouvent alors dans les deux catégories. La seconde distinction se fait sur les attributs de représentation. On catégorise souvent les techniques de visualisation

selon que la spatialisation est donnée ou choisie. Les auteurs ajoutent une catégorie intermédiaire lorsque la spatialisation est en partie choisie et en partie donnée. Par exemple, des positions géographiques sur un globe impliquent une représentation donnée. Pourtant, il reste une certaine marge de manœuvre dans la façon de les représenter sur un plan en deux dimensions, soit le choix de la projection. Ils ajoutent aussi que d'autres attributs que la spatialisation peuvent contraindre la représentation visuelle tels que le temps, les couleurs et la transparence. Ils spécifient que ce deuxième axe de catégorisation est en fait un continuum allant du totalement contraint au nullement contraint. Dans cette taxonomie, les techniques normalement classées comme de la visualisation scientifique se retrouvent typiquement vers les modèles continus aux attributs de représentation complètement contraints alors que les techniques normalement classées comme de la visualisation de l'information tendent à se retrouver vers les modèles discrets aux attributs de représentation non contraints. Les interfaces utilisateurs des applications auto-adaptatives seront ultimement appelées à présenter à la fois des données continues et des données discrètes, ainsi que des données contraintes et non contraintes.

Shneiderman (1996) propose de classer les techniques de visualisation de l'information non pas par type de données, mais par tâche. Les sept tâches qu'il propose sont la vue d'ensemble, le zoom, le filtre, le détail sur demande, la relation, l'historique et l'extraction. Il introduit dans ce même article ce qu'il nomme le mantra de la recherche visuelle d'information : "D'abord une vue générale, zoom et filtre, puis des détails sur demande (*Overview first, zoom and filter, then details-on-demand*)." Comme on le verra, ce mantra a inspiré d'autres chercheurs et est toujours d'actualité.

Liu, Cui, Wu & Liu (2014) font le lien entre l'état de l'art des recherches de pointes dans le domaine de la visualisation de l'information et leurs applications dans le monde réel. Les auteurs classent d'abord les travaux sur les techniques de représentation de l'information en quatre catégories : les méthodologies empiriques, les interactions, les cadres de développement et les applications.

Les travaux recensés dans la catégorie des méthodologies empiriques sont séparés en modèles et en évaluations selon qu'ils s'appliquent respectivement à plusieurs cas ou à un seul cas.

Parmi les modèles, on trouve des modèles de représentation visuelle aidant le passage de la recherche à la pratique, des modèles dirigés par les données issues d'applications du monde réel, et des méthodologies génériques permettant de guider le déploiement de techniques et d'outils de visualisation d'information. Des recherches sur la visualisation de l'incertitude des données y sont aussi présentées et celles-ci pourraient être particulièrement intéressantes dans le contexte du WS. Les évaluations sont quant à elles pour la plupart des méthodes d'évaluations basées sur des études d'utilisabilité et des expérimentations contrôlées. Elles permettent donc d'ancrer la mesure de la performance visuelle dans un cadre scientifique solide. Ces études utilisent des techniques des sciences humaines telles que les sondages, les études soumises à la foule ("*crowdsourcing*") et des études en laboratoires.

La catégorisation des interactions se divise en deux : les interactions dites WIMP pour *Windows, Icons, Mouse, Pointer* qui se réfèrent aux manipulations avec souris et clavier et les interactions dites post-WIMP qui se font à l'aide d'autres outils comme les doigts ou des crayons. Seule la première concerne la présente recherche. Les interactions WIMP comprennent des tâches comme la sélection, le filtrage, le surlignage, la comparaison, la navigation dirigée par les intérêts, la navigation dirigée par le point d'intérêt et la navigation dirigée par les facettes. Ces trois dernières sont particulièrement intéressantes à étudier pour la conception du mécanisme d'exploration ontologique.

La catégorie des cadres de développement recense les librairies et les trousses d'outils pour développer des mécanismes de visualisation. La catégorie des applications, quant à elle, présente différentes solutions à la visualisation des graphes, de textes, de cartes et de données multivariées.

Liu *et al.* (2014) décrivent ensuite les défis techniques à la représentation visuelle. Les cinq défis sont l'usabilité, les possibilités de mise à l'échelle visuelle, l'analyse intégrée de données hétérogènes, la visualisation *in-situ* ainsi que les erreurs et l'incertitude. Le défi de l'usabilité se réfère au manque de méthodes d'évaluation à la fois spécifiques à la visualisation et assez génériques pour évaluer un large éventail d'applications et de domaines reliés à la visualisation. Le défi de la mise à l'échelle visuelle concerne les difficultés à montrer visuellement soit un grand nombre de dimensions, soit un grand nombre d'individus. Des

techniques de réduction de données telles que l'échantillonnage, le filtrage, le groupement ou l'analyse en composantes principales ont un certain succès mais aucune n'est parfaite ni utilisable dans toutes les situations. Le défi de l'intégration des données hétérogènes concerne les données provenant de plusieurs sources et dans plusieurs formats. Le défi de la visualisation *in-situ* concerne la difficulté à visualiser les données en continue (*streaming)* tant au niveau de l'architecture logicielle que des techniques pour cerner l'information importante. Finalement, le défi des erreurs et de l'incertitude concerne l'inconsistance des données provenant du monde réel d'une part et l'incertitude qui change tout au long du processus de visualisation d'autre part. Ce défi est particulièrement important en ce qui concerne le WS. En effet, on retrouve dans Allemang & Hendler (2008) une discussion sur l'importance du slogan AAA ("*Anyone can say Anything about Any topic*") dans l'émergence du Web conventionnel et comment le WS doit pouvoir matérialiser cet adage tout en trouvant des moyens pour valider les données. Ces défis sont particulièrement intéressants dans le cadre de cette recherche et seront discutés au CHAPITRE 5.

1.2.5 Techniques de visualisation sémantique

Hachey et Gasevic (2011) offrent un bon point d'entrée dans l'étude de l'état de l'art des interfaces utilisateurs pour le WS en recensant et classant 87 articles publiés entre 2003 et 2011.

Dadzie et Rowe (2011) parlent de la nécessité de développer des solutions de visualisation sémantique permettant aux utilisateurs du Web sans savoir technique d'utiliser le Web de données liées. Ceux-ci doivent pouvoir comprendre la structure des données, construire des requêtes implicitement, identifier les liens entre les ressources et découvrir de nouvelles connaissances. Un problème qu'ils relèvent est la grande quantité d'information à laquelle l'utilisateur doit faire face. Tout d'abord, s'il est intéressé par une ressource, le nombre de liens de cette ressource vers d'autres ressources peut facilement mener à une surcharge informationnelle. De plus, la production à large échelle des données liées implique une perte de précision et de complétude de l'information. Cette perte de précision et cette incomplétude peuvent aussi entraîner des difficultés pour les utilisateurs ne connaissant pas le

fonctionnement de ces technologies. Finalement, si l'on désire que les néophytes sémantiques puissent tirer profit de cette technologie, il faut leur donner des outils leur permettant de construire des requêtes implicitement ou du moins, intuitivement.

Dadzie et Rowe recensent ensuite différentes techniques pour visualiser des données liées et expliquer les défis associés à ces techniques. Tout d'abord, ils abordent les représentations en graphe qui semblent naturelles sachant que les données RDF prennent *in naturalibus* la forme d'un graphe. Un premier défi vient de la force même de la technologie, les données étant fortement liées entre elles, la compréhension d'un graphe large et fortement interrelié devient difficile. De plus, des faux liens peuvent s'établir lors de la construction automatisée de ce genre de données et ceux-ci sont difficilement identifiables. En outre, des îlots se présentent souvent dans les graphes, représentant des poches de données qui ne sont pas liées au reste des données. Déterminer si leur présence est due à une erreur ou non est difficile. Comme les données liées sont par définition un effort communautaire, une hétérogénéité inhérente doit être envisagée. La fidélité et la fiabilité des données ne peuvent donc pas être garanties. Les différences dans les types de données, la qualité de la capture des données, le contenu, la granularité et la présence de concepts identiques sous des identifiants spécifiques sont différents aspects de ce problème.

Les auteurs discutent de l'importance de caractériser les utilisateurs de la technologie, leur compétence, leurs besoins et les tâches qu'ils doivent faire pour augmenter les chances d'adoption d'un outil. On sépare souvent les utilisateurs en trois catégories : débutant, intermédiaire et expert. Cependant, on peut aussi faire appel à la notion d' « utilisabilité universelle » (Shneiderman & Plaisant, 2010) pour les outils qui s'adressent à un large éventail d'utilisateurs. Les publics cibles des outils sont amplement discutés dans la littérature, en général comme un des éléments les plus importants dans la conception des interfaces.

Dadzie et Rowe vont par la suite définir une liste d'exigences qui devraient être satisfaites pour permettre la conception de techniques de visualisation sémantique efficaces. Ils vont séparer ces exigences en deux : celles que tout système de visualisation devrait adresser puis celles liées à la consommation des données liées. Dans le premier cas, à haut niveau, ils se réfèrent au *mantra de la recherche d'information visuelle* de Shneiderman (1996) : une vue générale

en premier, suivie de filtres et de possibilités de sélection, puis des détails sur demande. Ensuite, à plus bas niveau, ils ajoutent qu'un système de visualisation devrait permettre la manipulation de données multidimensionnelles, de données hiérarchiques (ou de structures en arbre), de données de réseaux (ou de structures en graphes), d'identifier et de surligner les relations entre les données et d'extraire des données.

Pour la consommation des données liées, en ce qui concernent les utilisateurs experts, un bon système de visualisation devrait permettre :

- Une navigation intuitive dans les structures de données liées;
- L'exploration des données pour saisir leur structure et leur contenu;
- L'exploration des données pour identifier les liens à travers le graphe, les erreurs et les anomalies;
- Des requêtes avancées qui utilisent une syntaxe formelle;
- La publication des données pour en permettre la vérification, la validation et l'extraction afin qu'elles puissent être réutilisées dans d'autres applications.

Dans le cas d'un système de visualisation de données liées conçu pour des utilisateurs débutants, ce système devrait permettre :

- Une navigation intuitive à travers un large graphe comprenant des données complexes et multidimensionnelles;
- Des possibilités de recherche exploratoire des connaissances;
- Le support pour des requêtes simples et avancées (filtrage et récupération d'information);
- Une analyse détaillée des régions d'intérêts;
- La publication des données;
- L'extraction de données;
- La présentation des résultats des analyses.

En se basant sur une revue des outils existants, les auteurs classent ces outils selon un sommaire de lignes directrices de conception et selon les fonctionnalités possibles de ces outils. Les lignes directrices de conceptions sont :

- Une présentation visuelle;
- Une vue d'ensemble des données;
- Des détails sur demande;
- Le surlignage des liens entre les données;
- Le support pour la mise à l'échelle;
- Le support pour les requêtes;
- Le filtrage;
- L'historique.

Les fonctionnalités sont :

- Des canevas de présentation permettant d'associer les types de données à des standards de présentation;
- Des points d'entrée dans les données liées, par une URI ou par des mots-clés ou encore grâce à une cartographie de celles-ci;
- Une généricité permettant à l'outil de parcourir des données de différents domaines
- La navigation par facettes pour supporter une recherche intuitive de données multidimensionnelles et mettre en évidence des relations cachées;
- La publication de nouvelles données;
- La fusion de différents jeux de données;
- L'indication de la provenance des données et de la confiance qu'on peut avoir en elles;
- L'édition des données pour pouvoir les enrichir et les corriger;
- La réutilisabilité des extrants.

Ces exigences, lignes directrices et fonctionnalités seront discutées au CHAPITRE 6.

1.2.6 Types de visualisation sémantique

Marie et Gandon (2014) classent les systèmes de visualisation sémantique en trois catégories : les navigateurs de données liées, les systèmes de recommandations sur les données liées et les systèmes de recherche exploratoire basées sur les données liées.

Vega-Gorgojo *et al.* (2016) dénombrent quatre types d'interfaces de recherches sémantiques dans la littérature : les éditeurs de requêtes, les interfaces basées sur la récupération de l'information, les interfaces en langage naturel et les approches visuelles. Les auteurs reconnaissent l'existence des navigateurs de données RDF mais déclarent que ceux-ci ne permettent que la navigation et non pas les requêtes. Cette classification diffère donc de celle de Marie et Gandon qui eux classent les trois dernières comme des formes de navigateurs sémantiques. Une des techniques de visualisation et d'exploration proposée dans la présente recherche transpose les éléments d'un navigateur dans le cadre d'un système de requêtes.

Soylu et Giese (2016) font eux tout d'abord la différence entre les paradigmes que sont la récupération de données (RD) (en anglais, *Data retrieval*) et la recherche d'information (RI) *(*en anglais*, Information retrieval*). Dans le paradigme RI, on tente de retrouver des objets pertinents en se basant sur une requête souvent incomplète et mal définie, grâce à un modèle probabiliste. Par opposition, dans le paradigme RD, la requête doit être exacte et complète et, grâce à un modèle déterministe, retrouver uniquement les objets correspondant à ses critères. Les applications de la présente recherche font parties du paradigme RD.

Selon Soylu et Giese (2016), c'est à l'intérieur du paradigme RD qu'on retrouve les systèmes de requêtes visuelles (SRV) (*Visual query systems*) et les langages de requêtes visuelles (LRV) (*Visual query language*). Un LRV représente chaque élément d'un langage de requête visuellement pour reproduire graphiquement une requête. Il ne simplifie donc pas l'utilisation du langage mais se borne à le représenter visuellement pour diminuer la charge mentale d'écriture d'une requête. Un SRV en revanche tente de simplifier le langage de requête pour protéger l'usager de sa complexité sous-jacente. Ainsi, le nombre d'actions possibles sera restreint, ce qui réduit l'expressivité du langage au profit de l'utilisabilité de l'outil et donc du temps que prendra un usager néophyte pour le maîtriser.

En effet, tel que mentionné dans Vega-Gorgojo *et al.* (2016), lorsqu'on compare différentes alternatives aux techniques de spécifications de requêtes pour la recherche sémantique, les deux critères qui intéressent sont l'expressivité et l'usabilité. Alors que l'expressivité représente la variété de requêtes qui peuvent être comprises par un système, l'usabilité, quant à elle, représente la facilité d'apprendre une interface par un utilisateur, son efficience et son

intuitivité. Les auteurs déclarent que le compromis entre ces deux dimensions amène régulièrement un sacrifice de l'expressivité d'un langage de requête au profit de l'utilisabilité d'un système.

Les quelques classifications présentées précédemment constituent un survol des paradigmes de représentations visuelles sémantiques qui n'est pas exhaustif. La quantité des classifications dans la littérature et la pléthore d'applications actuellement disponibles pouvant être classifiées rendent la construction d'une vision d'ensemble du domaine difficile à synthétiser. Cela est particulièrement ironique puisqu'une majorité des recherches sur les techniques de visualisation s'entendent sur le fait que la vue d'ensemble est précieuse pour se forger une compréhension d'un domaine, or ici c'est l'accumulation de visions à haut niveau qui entrave la construction d'un schéma mental global et cohérent.

Cela dit, les trois applications qui seront présentées dans les prochains chapitres peuvent entrer respectivement dans les catégories des navigateurs sémantiques, des systèmes de requêtes visuelles sémantiques et des constructeurs d'applications sémantiques.

1.2.7 Navigateurs sémantiques

Dadzie et Rowe (2011) font une revue des outils supportant la navigation à travers des données liées qu'ils nomment « les navigateurs de données liées ». Ils font une distinction entre ceux qui utilisent des représentations graphiques, picturales ou en images pour donner un instantané statique (visualisation de données liées) ou pour permettre une exploration interactive (navigateur visuel de données liées). Ils présenteront tout d'abord les navigateurs qui n'utilisent qu'une présentation et des fonctionnalités basées sur le texte et ensuite ceux qui permettent des options de présentation ou d'analyse visuelles. Les navigateurs basés sur le texte utilisent des canevas de présentation pour présenter les ressources dans des tables ou des listes et peuvent aussi afficher des images dont les URL sont contenues dans la base de connaissances.

Marie et Gandon (2014) distinguent quatre types de navigateurs de données liées : les textuels, les visuels, ceux qui fonctionnent par facettes et les navigateurs plus singuliers.

Plusieurs navigateurs sémantiques textuels existent. Marie et Gandon proposent de se référer à Dadzie, Rowe et Petrelli (2011) pour une liste complète. Ce type de navigateurs permet l'affichage des données sémantiques textuelles. Ils en citent un certain nombre qui donnent un tour d'horizon des différentes fonctionnalités dont Noadster (Noadster, 2020), Disco (Disco, 2020) et Marbles (Marbles, 2020) qui seront présentés dans le CHAPITRE 3.

Les navigateurs sémantiques visuels sont également nombreux et les auteurs réfèrent à Dadzie et Rowe (2011) pour une liste plus exhaustive. Ce type de navigateurs permet l'affichage de données sémantiques dans des paradigmes de visualisation plus complexes. Marie et Gandon recensent les fonctionnalités tirées de quelques applications choisies dont RDF Gravity (RDF Gravity, 2020), Tabulator (Berners-Lee, *et al.*, 2006), DBpedia Mobile (Becker & Bizer, 2008) et LENA (LENA, 2020) qui seront discutés au CHAPITRE 5.

Les navigateurs à facettes utilisent quant à eux les facettes pour partitionner l'espace de recherche. Celles-ci agissent comme des filtres; la sélection d'une facette permet de filtrer les résultats et les facettes disponibles sont alors mises à jour en fonction de ceux-ci. Les auteurs présentent encore quelques exemples de fonctionnalités de tels navigateurs. Longwell (Veres, Johansen, & Opdahl, 2010), MuseumFinland (Hyvönen, 2004), mSpace (Wilson, Russell, Schraefel, & Smith, 2006), \facet (Hildebrand, Van Ossenbruggen, & Hardman, 2006) et Humboldt (Kobilarov & Dickinson, 2008) seront présentés au CHAPITRE 5.

Parmi les autres paradigmes de navigation, on retrouve des solutions plus innovantes, telles que Parallax (Huynh & Karger, 2009), RelFinder (RelFinder, 2020), gFacet (Heim & Ziegler, 2009; Heim, Ertl, & Ziegler, 2010), et Visor (Popov, Schraefel, Hall, & Shadbolt, 2011). Elles seront aussi discutées au CHAPITRE 5.

Les systèmes de recommandation et de recherche exploratoire basée sur les données liées présentées dans Marie et Gandon (2014) sont en fait des versions plus achevées d'hybrides entre navigateurs sémantiques et systèmes de recommandations. Ils ne seront pas présentés en détails ici puisqu'à l'extérieur du cadre de cette recherche. Cependant, leur potentiel serait à explorer pour bonifier les solutions créées. Les systèmes de recommandation se basent pour la plupart sur la similarité sémantique et se divisent en approches où le domaine est contraint (qui se concentrent sur un sous-ensemble d'un jeu de données) et en approches inter-domaines. Les

systèmes de recherche exploratoire basée sur les données liées, quant à eux, utilisent des fonctionnalités dédiées à l'exploration et sont divisés en deux sous-classes : ceux permettant de produire des vues sur des graphes grâce à des opérateurs avancés et ceux appliquant des algorithmes sur les données sémantiques pour sélectionner et ordonner, ne présentant que les données et leurs propriétés qui sont d'intérêt.

1.2.8 Systèmes de requêtes visuelles

Les constructeurs de requêtes visuelles (CRV) sont de bons exemples de paradigmes de visualisation pouvant s'appliquer à la fois dans des systèmes d'information conventionnels et dans des systèmes d'information se basant sur des données sémantiques. Les BDR ont des CRV tels qu'Access (Access, 2020) ou Toad (Toad World, 2020). Ces outils font partie des vecteurs d'adoption de ces technologies puisqu'ils permettent de rendre accessible l'utilisation des bases de données aux personnes n'en connaissant pas le langage de requête.

Un CRV doit supporter deux activités complémentaires mais opposées : l'exploration et la construction. Une préoccupation majeure est donc de choisir des représentations visuelles et des paradigmes d'interaction appropriés pour supporter ces deux activités (Soylu & Giese, 2016). Les représentations visuelles les plus utilisées sont : la représentation en graphe, la représentation par formulaire et la représentation par facettes (Soylu & Giese, 2016). Les représentations en graphes permettent généralement de construire des requêtes en ajoutant itérativement des nœuds et des arcs au graphe. L'utilisation de cette seule technique de représentation visuelle peut cependant rendre l'interface visuellement complexe à comprendre. La littérature est d'ailleurs de plus en plus critique face à l'utilisation *de facto* des graphes pour représenter les données sémantiques (Dadzie & Pietriga, 2017; Karger, 2006). La représentation par formulaire utilise quant à elle des boîtes de textes, des boutons de sélection et des menus déroulants pour filtrer l'espace de solution. Finalement, la représentation par facettes, qui peut être vue comme une sous-classe de la représentation par formulaire, se base sur le vocabulaire du domaine pour restreindre l'espace de solution en filtrant celui-ci grâce aux différentes facettes des concepts (Vega-Gorgojo, *et al.*, 2016). Les facettes peuvent être définies comme des dimensions conceptuelles orthogonales partitionnant l'espace de

recherche (Heim, Ertl, & Ziegler, 2010). Les facettes peuvent être vues comme des étiquettes de métadonnées appliquées sur les données qui permettent d'être sélectionnées afin de réduire l'espace de recherche. Les triplets peuvent s'utiliser comme des facettes, les classes, les propriétés ou mêmes les individus pouvant être utilisés pour réduire l'espace de solution des requêtes.

Afin de pouvoir comparer les fonctionnalités des CRV sémantiques avec les CRV relationnelles, il convient de connaître ces derniers.

Toad fut tout d'abord un SGBD créé vers la fin des années 1990 par Jim McDaniel (Toad (logiciel), 2020). Aujourd'hui, Quest Software est propriétaire de Toad World qui offre plusieurs produits dont des outils d'aide au développement et à la gestion de bases de données pour plusieurs systèmes tels que MySQL, IBM/DB2, SAP, SQL Server et Oracle. Le CRV de Toad se décline en plusieurs outils spécialisés tels que Toad for Oracle (Toad for Oracle, 2020) et Toad Data Point (Toad Data Point, 2020). Toad for Oracle permet d'explorer visuellement des bases de données relationnelles Oracle, d'automatiser des flux de travail, d'effectuer les tâches essentielles de développement et d'administration, de valider et d'optimiser la BDR ainsi que de gérer et partager des projets, gabarits et scripts. Toad Data Point permet quant à lui de se connecter à plusieurs types de BDR, de faire des requêtes sur plusieurs sources simultanément, de préparer la donnée et de l'exporter.

Microsoft Access est un SGBD de la suite Microsoft Office supporté par le langage Visual Basic for Applications (VBA) et qui peut être utilisé pour construire des applications logicielles. Il peut aussi faire office d'interface utilisateur. Il fut originellement conçu pour permettre aux utilisateurs finaux d'accéder à des données de n'importe quelle source (Microsoft Access, 2020). Il offre lui aussi un CRV SQL intégré.

Les CRV SQL de Toad et d'Access utilisent les trois types de visualisation mentionnés précédemment : une vue en graphe des classes (tables) et de leurs relations (jointures), une vue en facettes des propriétés (attributs) des classes et une vue en formulaire pour appliquer des filtres sur les propriétés. Dans les deux cas, une liste permet de choisir les classes, qu'elles soient réelles (tables de la BDR) ou virtuelles (vues ou synonymes). Ces classes sont glissées et déposées dans un espace de travail graphique où la requête visuelle est construite. Dans cet

espace, les classes sont représentées par des boîtes contenant la liste des propriétés reliées aux classes qui peuvent être sélectionnées pour être ajoutées à la requête (vue en facettes). De plus, les deux outils permettent d'ajouter des clauses (filtres) sur les requêtes grâce à des vues en formulaire.

Ce sont donc ces paradigmes visuels qui ont été sélectionnés comme point de départ pour le développement du CRV SPARQL d'HQ. Dans le CHAPITRE 4, les fonctionnalités spécifiques du CRV SQL de Toad (Toad for Oracle, 2020) ainsi que des CRV SPARQL suivants seront présentées : OptiqueVQS (Soylu, *et al.*, 2013; Soylu, *et al.*, 2013; Ahmetaj, Calvanese, Ortiz, & Šimkus, 2014; Bagosi, *et al.*, 2014; Soylu A. , *et al.*, 2014; Giese, *et al.*, 2015; Soylu A. , *et al.*, 2015; Soylu & Giese, 2016); Facet Graphs (gFacet) (Heim & Ziegler, 2009; Heim, Ertl, & Ziegler, 2010); Mash-QL (Jarrar & Dikaiakos, 2008; Jarrar & Dikaiakos, 2008; Jarrar & Dikaiakos, 2009; Jarrar & Dikaiakos, 2011; Medhe & Phalke, 2012); SparqlFilterFlow (Haag, Lohmann, & Ertl, 2014; Haag, Lohmann, Bold, & Ertl, 2014); GRQL (Athanasis, Christophides, & Kotzinos, 2004); Konduit (Ambrus, Möller, & Handschuh, 2010) et SPARQLGraph (Schweiger, Trajanoski, & Pabinger, 2014; Schweiger, 2014; Belleau, Nolin, Tourigny, Rigault, & Morissette, 2008). Une pléthore d'autres CRV SPARQL ont été recensés, mais ils ne feront pas l'objet d'une étude plus approfondie.

1.2.9 Constructeurs d'applications sémantiques

Tel que vue précédemment la construction automatique d'applications est un vieux rêve que portent encore l'IDM et l'IDO. Quelques solutions commerciales existent pour tenter d'harnacher l'évolutivité et la flexibilité des technologies sémantiques afin de construire automatiquement des applications.

TopQuadrant (TopQuadrant, 2020) offre toute une panoplie de produits utilisant les technologies sémantiques. Parmi celles-ci, la suite TopBraid (TopBraid, 2020) implémente le *SPARQL Inferencing Notation* (SPIN) (SPIN, 2020), un standard pour représenter des règles et contraintes SPARQL qui comporte des capacités de méta-modélisation permettant d'intégrer des fonctions SPARQL et des canevas de requêtes à l'intérieur des modèles sémantiques. La pile technologique de SPIN (SPIN Technology Stack, 2020) est étendue par *SPARQL Web*

Pages (SWP) (SWP, 2020), un cadre de développement basé sur les RDF pour décrire des interfaces utilisateurs permettant la visualisation des données sémantiques. L'avantage de SWP est de permettre d'intégrer à la fois des requêtes SPARQL dynamiques et la façon de présenter les résultats sous forme de code *Hypertext Markup Language* (HTML), *Scalable Vector Graphics* (SVG), *JavaScript Object Notation* (JSON) ou XML, à l'intérieur même du modèle de données.

De son côté, inova8 (inova8, 2020) tente d'aider l'utilisation des données liées dans les entreprises en offrant des solutions logicielles facilitant les échanges entre les différentes composantes logicielles. On retrouve dans leur suite OData2SPARQL, un protocole pour créer et consommer des interfaces de programmation (en anglais *Application Programming Interface*, API) bâtissant sur le protocole de transfert hypertexte (en anglais, *Hypertext Transfer Protocol,* HTTP) et sur la méthodologie REST (en anglais, *Representational state transfer*). inova8 propose d'utiliser leurs technologies avec la librairie OpenUI5 (OpenUI5, 2020) pour la création rapide d'applications.

Ces deux solutions seront discutées plus en détails au CHAPITRE 5.

1.3 Contexte industriel

Les présents travaux ont été réalisés pour le compte d'Hydro-Québec et il convient donc maintenant d'introduire le contexte industriel de ceux-ci. Tout d'abord, le contexte global de l'Industrie 4.0 sera abordé avant de traiter du contexte spécifique de cette entreprise.

1.3.1 Industrie 4.0

Le monde se trouve actuellement au début de la quatrième révolution industrielle : l'Industrie 4.0. Ce concept peut être vue comme une nouvelle méthode d'organisation des moyens de production tablant sur l'intégration des outils numériques et du monde réel. Tous les équipements de la chaîne de production devenant instrumentés, il devient en théorie possible de l'optimiser de façon holistique et autonome.

Cette révolution est fondée sur l'intégration des données qu'a entraînées la multiplication des senseurs à toutes les étapes de la chaîne logistique et sur l'interconnectivité des machines dans les systèmes cybernétiques (Lee, Bagheri, & Kao, 2015). L'Industrie 4.0 peut être définie comme un ensemble de technologies qui transforment la façon dont on gère les systèmes connectant les actifs physiques aux capacités de calculs.

Elle repose sur neuf piliers correspondant à ces technologies qui permettent de connecter et d'analyser les données, de prédire les états futurs du système et de reconfigurer les composants de la chaîne de valeur (Lee, Bagheri, & Kao, 2015). Parmi ces technologies, deux sont pertinentes à la présente recherche : l'internet industriel des objets (en anglais *Industrial Internet of Things*, IIoT) et l'intégration horizontale et verticale des systèmes.

Le pilier de l'intégration des systèmes implique que tous les systèmes de l'organisation sont interconnectés et que les organisations le sont entre elles également. La technologie de l'internet des objets (en anglais *Internet of Things*, IoT), quant à elle, fait référence à un réseau d'objets interconnectés. L'IIoT est l'application de l'IoT dans un contexte industriel. Grâce à cette interconnexion les différents composants de la chaîne de valeur peuvent interagir entre eux et avec l'environnement, facilitant ainsi leur adaptation rapide aux changements (Vaidyaa, Ambadb, & Bhoslec, 2018).

Le Web sémantique des objets (en anglais *Semantic Web of Things*, SWoT) est étudié comme une des avenues prometteuses pouvant faciliter l'interopérabilité de tels systèmes en permettant l'intégration d'objets hétérogènes grâce à une information sémantiquement riche (Vogel-Heuser & Hess, 2016). Le SWoT est en quelque sorte l'intégration des technologies du Web sémantique avec l'IoT (Jara, *et al.*, 2014).

Différents besoins de l'Industrie 4.0 peuvent être abordés par les technologies du WS. Par exemples, le déploiement rapide d'applications, le développement d'approches holistiques de réingénierie des systèmes et la possibilité d'utilisation des applications par des utilisateurs néophytes des technologies sous-jacentes (Thuluva, Anicic, & Rudolph, 2017). Chen *et al.* (2017) proposent l'utilisation des ontologies dans les usines du futur en raison de leurs propriétés d'auto-organisation, d'auto-apprentissage et d'auto-adaptativité.

Alors que l'IoT se veut un réseau d'objets interconnectés, le Web des objets (en anglais *Web of Things*, WoT) vise à intégrer ces objets au niveau du Web. Le SWoT vise quant à lui à tirer profit du WS et du WoT afin de faciliter la création d'un réseau de connaissances en ajoutant de la signification aux données et en facilitant l'interopérabilité des systèmes cybernétiques (Wu, *et al.*, 2017).

Il serait souhaitable que les applications informatiques de l'Industrie 4.0 puisant dans le SWoT puissent s'adapter automatiquement aux modèles de données des objets. Deux concepts clés sont importants pour atteindre ce genre de flexibilité : la généricité de l'application et son auto-adaptabilité au modèle de données.

La généricité est ici définie comme la capacité de l'application à fonctionner avec des connaissances provenant de n'importe quel domaine. L'auto-adaptabilité, quant à elle, est définie comme étant la capacité de l'application à s'ajuster au contenu du modèle de données lorsque des changements sont apportés à l'ontologie, sans nécessiter de modification ou de recompilation du code. Les applications présentées dans cette recherche mettent en œuvres des principes logiciels tels que l'indépendance conceptuelle qui pourrait jouer un rôle prépondérant dans la transition vers l'Industrie 4.0.

1.3.2 Systèmes d'information chez Hydro-Québec

Depuis au moins une quinzaine d'années, des recherches ont été effectuées à l'IREQ dans le domaine des applications auto-adaptatives avec, entre autres, l'environnement de développement de Modélisation dynamique de l'information (MDI) (Vouligny & Robert, 2005). Certaines applications client-serveur construites avec ce système ont été mises en production et sont toujours utilisées aujourd'hui. L'auto-adaptativité, même si elle ne se fait uniquement qu'au niveau du modèle de données, s'est montrée bénéfique, spécialement lorsque le prototypage évolutif était utilisé comme méthodologie de développement (Vouligny, Hudon, & Nguyen, 2009). Dans MDI, la librairie de développement proposée n'était pas un cadre de développement client-serveur mais était utilisée en tant que système de modélisation sémantique privé et fermé. Les avantages tirés de MDI ont motivé la continuation de ces

recherches en utilisant des standards tels que RDF, rendant possible un développement de concert avec d'autres chercheurs de la communauté internationale.

Des recherches sur les technologies du Web sémantique ont alors suivi, notamment comme un moyen de permettre l'interopérabilité entre les toujours plus nombreux systèmes d'information de l'entreprise (Gaha, *et al.*, 2013; Zinflou, *et al.*, 2013). Certaines recherches ont permis d'effectuer des requêtes fédérées sur des modèles sémantiques et des historiens (Gaha, *et al.*, 2015).

HQ est le principal producteur d'électricité au Canada avec 63 centrales hydro-électriques et une puissance installée de plus de 37 000 mégawatts. Elle comptait environ 4,3 millions de clients en 2018. HQ possède un très grand nombre de systèmes d'entreprise. Pour donner un ordre de grandeur, le système de conduite du réseau se base à lui seul sur plus de 80 systèmes d'entreprise.

Un exemple de système d'HQ est le Système de commande et de surveillance en centrale (SCSC) qui donne accès aux données de surveillance des groupes turbines-alternateurs (GTA). Différents outils se servent du SCSC pour analyser le comportement des GTA et des systèmes auxiliaires, pour repérer des anomalies, poser des diagnostics, aider à la mise en route et en service des GTA, aider aux maintenances systématiques, conditionnelles et prédictives, etc.

Le Système de contrôle et d'acquisition de données (SCADA) est un autre exemple notable de système d'entreprise. Il sert à évaluer les ressources en eau disponibles pour les centrales. Il est constitué de plusieurs centaines de capteurs de niveau d'eau (limnimètres), d'ouverture de vanne, de tension, de puissance active et réactive, de débit, etc. Il est lui aussi à la source de plusieurs outils d'analyses et de prévisions de l'entreprise dont le système de planification de la production hydroélectrique.

La première application qui a été développée dans le cadre de cette recherche, DATE, puise ses données dans un système appelé « Étude du vieillissement de l'appareillage (EVA) ». Ce système vise à assister les électriciens dans leurs tâches quotidiennes de vérification et d'entretien de l'appareillage électrique en leur permettant notamment de consigner dans une base de données les résultats des essaies qu'ils effectuent. Plusieurs outils informatiques puisent leur information dans EVA afin de contribuer à l'optimisation des processus de

maintenance des actifs de transport d'énergie. EVA permet, entre autres, d'effectuer des calculs statistiques sur certains types d'essais, de repérer les anomalies et d'assurer un suivi sur celles-ci, ainsi qu'à visualiser, normaliser et exporter les données.

EVA interagit avec d'autres systèmes d'entreprise dont SAP qui contient l'inventaire de tous les appareillages d'HQ. EVA communique avec 75 serveurs auxquels environ 800 utilisateurs se connectent pour leurs travaux. EVA est constituée de plusieurs BDR décentralisées qui sont de trois types : des BDR opérationnelles contenant les données d'essais électriques, des BDR territoriales contenant les essais sur les huiles et une base de données provinciale contenant les normes et les agrégations des BDR territoriales.

Dans le cadre d'un projet de maintenance, un groupe de chercheurs de l'IREQ a produit des formules permettant de caractériser l'état de dégradation de l'isolation des transformateurs grâce à la présence de composés chimiques dans l'huile. Des protocoles de mesures de présence de ces composés ont été élaborés et mis en œuvre sur le terrain. Les données de ces tests d'huile de transformateurs sont désormais accessibles par le système EVA. Ce sont ces données qui sont utilisées par l'application DATE.

Afin d'assurer la continuité de sa compétitivité à l'internationale, une grande entreprise ayant autant de systèmes d'information qu'HQ a tout intérêt à déjà prévoir sa transition vers l'Industrie 4.0. Les technologies du Web sémantique semblent pouvoir l'aider à s'y rendre.

CHAPITRE 2

MÉTHODOLOGIE

Tel que mentionné précédemment, cette recherche a été réalisée dans les laboratoires de l'IREQ et chaque étape de celle-ci correspond à un besoin d'affaires de l'entreprise. Chacun de ces besoins d'affaires a été abordé par le développement d'une application Web.

2.1 Questions de recherche

Les trois applications présentées dans les prochains chapitres sont autant de tentatives de répondre aux deux mêmes questions de recherche :

- Comment peut-on mettre à profit les technologies du WS afin de créer des applications Web s'auto-adaptant aux changements du modèle de données?
- Comment peut-on rendre ces applications manœuvrables par des utilisateurs débutants dans un contexte industriel?

2.2 Hypothèses

Ainsi, plusieurs hypothèses ont été émises afin de répondre à ces questions de progresser vers l'atteinte des objectifs présentés à la section 0.2 de l'Introduction.

La première hypothèse posée et dont la validité a dû être testée de prime abord énonçe qu'il est possible d'atteindre l'auto-adaptativité par l'indépendance conceptuelle. Elle a été confirmée par la réalisation de la première application, DATE, qui sera présentée au CHAPITRE 3.

Afin de développer cette application, différentes pistes de conception semblaient envisageables et c'est ainsi que l'hypothèse stipulant que l'utilisation d'un véhicule générique pouvait permettre d'atteindre les différents objectifs a été émise. Le fait que ce véhicule mis en place dans la première application ait été adapté et ré-utilisé dans les deux applications subséquentes confirme le bien-fondé de son utilisation.

Dans le même ordre d'idée, l'hypothèse que des composantes logicielles génériques jumelées à des requêtes génériques pouvaient permettre la création d'application Web auto-adaptatives et utilisables par des néophytes se confirme un peu plus à chaque application développée.

Finalement, dans le cas du CRV-HQ, la principale hypothèse étant à la base de son développement est que l'utilisation de paradigmes de visualisation et d'interactions simples et connus facilite l'adoption d'un nouvel outil, d'une nouvelle technologie. Ce CRV sémantique a donc été conçu afin d'offrir les mêmes fonctionnalités que les CRV relationnels, encapsulées dans les mêmes paradigmes d'interactions. Cette hypothèse sera discutée plus en détail aux CHAPITRE 4 et CHAPITRE 6.

2.3 Étapes de conception et élaboration

La méthodologie de conception pour la réalisation de chacune des trois applications a été sensiblement la même. Les différents projets ont débuté par des rencontres avec les clients pour définir leurs besoins. Des séances ont été organisées pour expliquer aux clients les spécificités de la présente recherche et les implications qu'elles auraient sur les produits en résultant.

Ensuite des designs préliminaires ont été conçus et discutés avec les membres du projet de support à la recherche en intelligence artificielle de l'IREQ. Ce projet horizontal regroupe un panel de chercheurs dont Luc Vouligny, le co-directeur de cette thèse, est l'instigateur. Ces chercheurs ont collaboré sur plusieurs des projets de l'IREQ présentés dans le chapitre précédent et ils ont apporté une vision plus juste et de très bonnes idées sur la conception des différents outils.

Après la présentation des designs préliminaires aux clients et l'obtention de leur accord, les projets ont été conçus de manière itérative en adoptant les principes de bases du développement agile. De courtes itérations de développement suivies de validations auprès du client se sont succédé jusqu' à l'obtention des produits finaux.

Les deux premières applications ont été installées dans les serveurs internes de l'IREQ. La troisième application a été mise en production dans les systèmes d'entreprise d'HQ et est

maintenant accessible à partir de l'intranet de l'entreprise. Les étapes et enjeux de mises en production sont discutés au CHAPITRE 6.

Plusieurs stagiaires universitaires du niveau baccalauréat ont travaillé au développement de ces applications. Leurs contributions font sans contredit partie des éléments clefs de réussite de cette recherche. En tout, sept étudiants se sont impliqués au cours de 12 sessions sur les différents projets issus de ces recherches. Ils ont contribué à la conception, au développement, à la validation et à la documentation des applications présentées dans les prochains chapitres. Plusieurs chercheurs, programmeurs, ingénieurs, techniciens et gestionnaires ont aussi apporté du support aux développements de ces applications et ont ainsi contribué à la réussite de leur implémentation.

Les différentes difficultés entraînées par l'utilisation des technologies du WS pour le développement d'applications auto-adaptatives ont été résolues au fur et à mesure qu'elles se sont présentées. Peu présents dans la littérature, de tels problèmes ont été surmontés par la réalisation de multiples essais et expériences. L'orientation appliquée de cette recherche et les contraintes de temps pour la livraison des produits n'a laissé que peu de place à la documentation des expériences en tant que telle, d'autant plus que plusieurs de celles-ci se sont avérées infructueuses ou finalement peu intéressantes. Ainsi, peu de chiffres permettant des analyses quantitatives sur des éléments précis ont subsistés. Seules les solutions retenues ont été conservées et elles sont désormais intégrées à l'une ou l'autre des trois applications qui sont présentées. Cela dit, plusieurs apprentissages ont été réalisés et ceux-ci sont colligés dans les prochains chapitres.

2.4 Pile technologique

Les trois applications ont été conçues en utilisant sensiblement les mêmes technologies logicielles. Elles ont été conceptualisées comme des applications trois-tiers, i.e., une interface Web communiquant avec un serveur générique qui interroge une BDT (voir Figure 2.1).

Les interfaces clients sont programmées en JavaScript (Javascript, 2020) grâce au cadre de développement Ext JS de Sencha (Sencha Ext JS, 2020). À l'époque du commencement de

cette recherche les cadres de développement Web ont été étudiés et deux d'entre eux semblaient plus appropriés pour bâtir des applications internet riche (Rich Internet application, 2020) : Angular (Angular, 2020) et Ext JS. Puisque du matériel avait déjà été développé avec Ext JS, ce cadre a été sélectionné. Le cadre de développement s'est avéré performant pour faire ce travail, mais sachant qu'Angular est devenu quelques années plus tard le cadre préconisé par l'entreprise, force est d'avouer que le hasard ne nous a pas été favorable. Le choix d'un cadre particulier oriente une partie du développement, chacun ayant des spécificités de programmation qui font qu'il devient difficile de convertir une solution vers un autre cadre.

La communication entre le client et le serveur se fait grâce à des services de type REST (REST, 2020). Les services de la première application sont contenus dans des *Enterprise JavaBeans* (EJB, 2020) sur un serveur TomEE (TomEE, 2020). La deuxième et la troisième application utilisent plutôt le cadre de développement Spring (Spring, 2020) pour s'occuper de la gestion des services et des droits des utilisateurs et sont déployées sur un serveur TomCat 7 (Tomcat, 2020). Pour les trois applications, le côté serveur est développé en Java 8 (Java, 2020). Les choix furent faits afin d'aligner le développement de ces applications avec les directives des groupes TI de l'entreprise aux époques où les applications ont été développées.

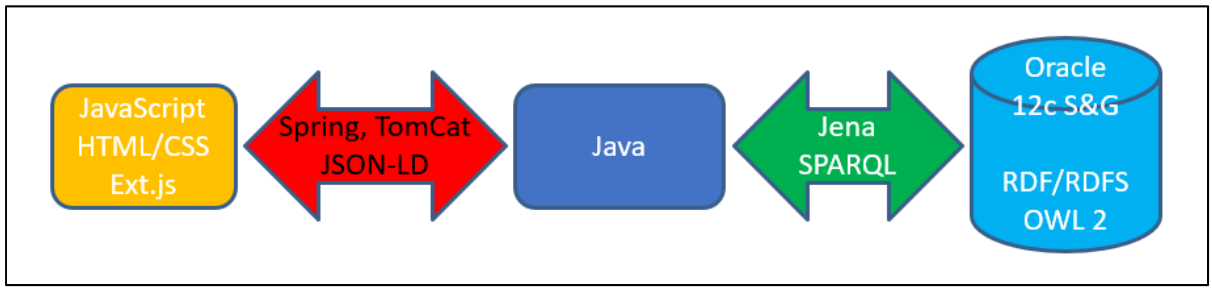

Figure 2.1 Pile technologique des applications sémantiques auto-adaptatives

Le côté serveur communique avec des bases de données en graphe RDF s'appuyant sur la version 12c d'Oracle (Oracle, 2020) par un module spécialisé : *Spatial and Graph* (Spatial and Graph, 2020). Des requêtes SPARQL sont utilisées par l'entremise de la librairie Jena (Jena, 2020). L'entreprise ayant une large base de produits Oracle et le module *Spatial and Graph*

étant déjà déployé dans des serveurs de l'entreprise, ce choix s'est imposé. La librairie Jena est la seule librairie toujours développée et maintenue pour lier Oracle et Java.

2.5 Plan des prochains chapitres

Les trois prochains chapitres présentent les trois applications développées pendant la recherche. Ces chapitres ont tous les trois la même structure. Tout d'abord est présenté le contexte industriel et le besoin d'affaires à lequel devait répondre l'application. Ensuite, l'application est présentée et les différents enjeux de conception sont décrits plus ou moins en détails en fonction de leur importance. Finalement, les chapitres sont conclus par une courte récapitulation des éléments importants.

Les discussions présentant les différents enjeux ayant émanés de l'implémentation des solutions sont discutées au CHAPITRE 6. Ces enjeux sont séparés en quatre catégories : la comparaison des fonctionnalités de la solution avec d'autres solutions ou cadres de développement, les problèmes techniques abordés pendant l'implémentation, les réflexions théoriques issues de la solution et les leçons apprises pendant la recherche. Chacune de ces catégories est séparée en trois sections, une pour chacune des applications développées.

CHAPITRE 3

DONNÉES D'ANALYSE DES TRANSFORMATEURS ÉLECTRIQUES

Puisque les connaissances d'une entreprise évoluent dans le temps, que ce soit par l'expérience, la recherche ou l'évolution de son environnement, l'utilisation de ces connaissances dans les systèmes de l'entreprise est aussi appelée à évoluer. Cette évolution est entravée par plusieurs facteurs logiciels dont la rigidité du modèle de données qui doit la plupart du temps garder sa forme originale pour que les systèmes qu'il alimente continuent à fonctionner correctement. Tout changement au modèle entraîne donc des opérations d'évolution et de maintenance sur toutes les applications l'utilisant.

Les technologies du WS ont une caractéristique qui vient fondamentalement changer la donne par rapport aux BDR traditionnelles. Les données, le modèle de ces données et les standards de méta-modélisation (RDFS, OWL) sont tous persistés dans le même langage : RDF.

Cette caractéristique peut sembler de prime abord futile. Cependant, c'est elle qui a permis de développer des applications dont les fonctions de lecture et de visualisation s'adaptent virtuellement à n'importe quel modèle suivant le ou les standards de méta-modélisation choisis. S'adaptant inhéremment à n'importe quel modèle, l'application auto-adaptative s'adapte aussi inhéremment aux modifications de ce modèle et libère ainsi le modèle de données de ses chaînes applicatives. C'est un double avantage qui est englobé dans le concept d'auto-adaptivité : d'une part les applications et leurs fonctionnalités s'adaptent à de nouveaux modèles; d'autre part, les modèles peuvent évoluer à leur guise.

C'est devant ce constat qu'il a été décidé de construire un premier cadre de développement permettant d'utiliser n'importe quel modèle de données pour bâtir avec un minimum d'effort un système client-serveur auto-adaptatif présentant les données à l'usager et lui permettant les opérations CRUD sur ces dernières.

Le présent chapitre portera sur DATE, une application puisant ses données d'un modèle sémantique. Son but était de démontrer la faisabilité d'une application s'adaptant dynamiquement à certains changements du modèle de données sans nécessiter de modification

ni de recompilation des couches client et serveur. Les recherches contenues dans ce chapitre ont été publiées dans Bhérer, Vouligny, Gaha, Redouane & Desrosiers (2015) et Bhérer, Vouligny, Gaha & Desrosiers (2016) et certaines parties de ces articles sont reproduites en traduction française dans ce qui suit.

3.1 Contexte

La première preuve de concept d'application auto-adaptative réalisée dans le cadre de cette recherche s'appelle DATE, pour « Données d'Analyse des Transformateurs Électriques ». Cette application a été réalisée pour répondre aux besoins d'un chercheur en chimie de l'IREQ, M. Jocelyn Jalbert, pour faire le suivi de ses travaux de caractérisation de la détérioration des transformateurs de puissances d'HQ. Il s'agit d''une application pour surveiller la santé de ces instruments à partir des concentrations de certaines molécules présentes dans leur huile, dont l'éthanol et le méthanol. Ces mesures servent à évaluer la condition de santé de ces coûteuses machines et donc aussi à assister les spécialistes dans la prévision des opérations de maintenance nécessaires.

C'est grâce à ces mesures que M. Jalbert a développé des formules permettant de déterminer l'état de détérioration des isolants des transformateurs de puissance. Cette façon de procéder s'est avérée une alternative rentable aux dispendieuses inspections par prélèvement d'échantillons d'isolant qui impliquent des interventions dans les transformateurs nécessitant l'ouverture de ceux-ci. Ces mesures sont recueillies par des techniciens sur le terrain qui les enregistrent dans le système d'acquisition EVA.

L'application agit en tant que tableau de bord duquel les utilisateurs peuvent ajouter, changer ou effacer des entrées et effectuer des recherches sur celles-ci. Elle sert aussi à effectuer certains calculs automatiques tels que l'ajustement des concentrations des molécules en fonction de la température de l'huile. Les ingénieurs utilisent en outre cette application pour enregistrer des opérations de maintenance et des mesures qui ne se retrouvent pas dans EVA ainsi que pour tester et raffiner les paramètres utilisés dans les équations d'ajustement des concentrations moléculaires.

3.2 Solution développée

Cette application semblait être un cas idéal pour débuter les expérimentations avec ces technologies puisqu'elle était simple mais nécessitait toutefois certains des composants de base dont l'exploration de l'implémentation était souhaitée, tels que des arbres, des grilles et des graphiques. Au cours de cette étape de la recherche des enjeux fondamentaux du cadre de développement ont été étudiés. Parmi ceux-ci : la communication client-serveur par véhicule générique, les différentes techniques de représentation visuelle de l'information, l'implémentation des services CRUD avec les technologies sémantiques et la construction dynamique de graphiques.

3.2.1 Vue d'ensemble

Afin de mieux comprendre le fonctionnement de l'application, la Figure 3.1 présente une vue de l'interface utilisateur (en anglais *user interface*, UI). À la phase d'initialisation, l'application interroge la base de données de triplets via un service Web. Cette première requête permet d'afficher la présentation de départ : une vue en arbre du modèle de données. Les utilisateurs se servent cet arbre pour sélectionner un individu d'une classe, e.g. un transformateur de puissance, représenté par une feuille de l'arbre. Quand l'utilisateur choisit une feuille, l'interface envoi une requête au serveur au travers des services Web. Sur réception de cette requête, le serveur récupère dynamiquement un nombre *a priori* inconnu de classes, ces dernières ayant une relation d'association avec la classe de l'individu sélectionné. Pour chacune de ces classes, le serveur récupère la liste de ses propriétés et la liste de ses individus ayant une relation avec l'individu sélectionné par l'utilisateur.

Ces informations transitent jusqu'au client sous la forme d'un objet générique afin d'établir la présentation des différentes composantes du client Web.

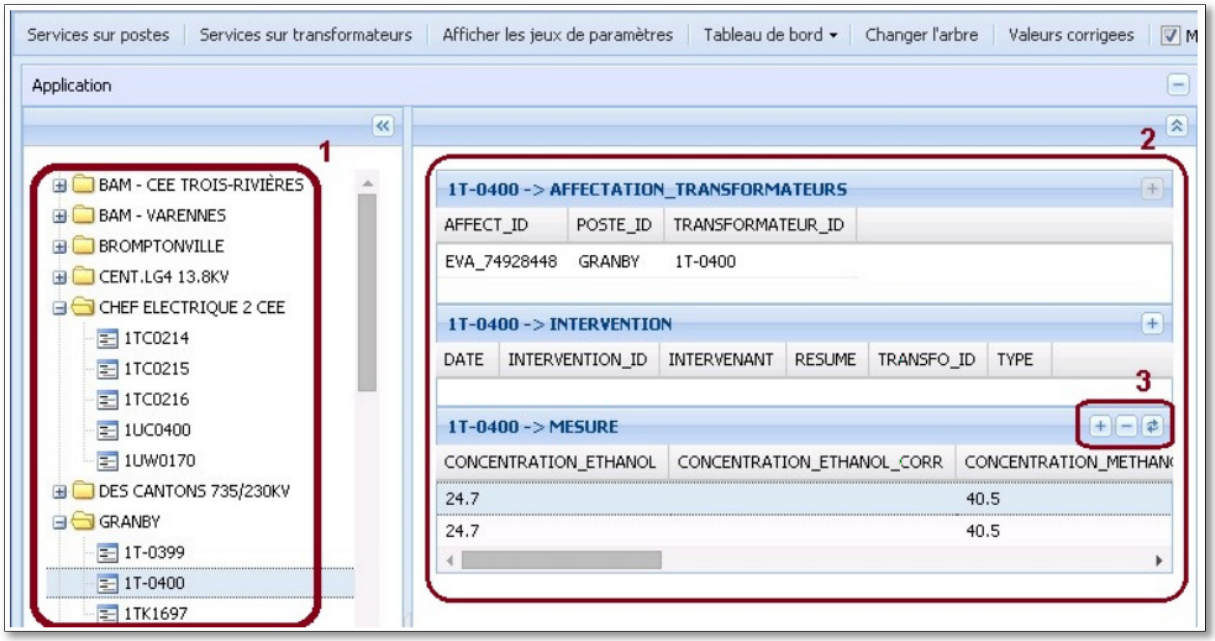

Figure 3.1 Vue d'ensemble de DATE

La vue en arbre (1), les matrices (2) et les boutons des formulaires CRUD (3)

Le modèle de données de cette première application est très simple et ne contient que six classes. Celles-ci seront utilisées dans les exemples présentés dans ce chapitre, soit :

1) Les postes électriques,
2) Les transformateurs de puissance,
3) Les interventions sur les transformateurs,
4) Les mesures sur les transformateurs,
5) Les paramètres de conversion des mesures,
6) Les affectations des transformateurs aux postes.

Chacune de ces classes contient entre 2 et 12 propriétés et comprend jusqu'à 7000 individus. Outre le fait que cette application requérait une gamme de fonctionnalités adaptées pour une large variété d'applications, elle a aussi été choisie comme banc d'essai des technologies du WS puisqu'elle présentait une ontologie simple, facile à construire et à tester.

Le modèle de données de l'application a été construit suivant le standard RDFS et les données proprement dites ont été importées depuis une BDR, comme il sera présenté dans la section suivante.

3.2.2 Importation des données et modèle de données

Les données brutes étant contenues dans une vaste BDR, celle-ci a tout d'abord été explorée et les tables et colonnes de données nécessaires à l'application ont été sélectionnées manuellement. À partir de celles-ci, un modèle sémantique pour héberger ces données a été conçu, puis les données ont été extraites et transformées en triplets RDF grâce à la librairie D2RQ. Le graphe formé par les données importées d'une BDR en est un d'individus (ABox) qui doit être décrit par un modèle formel (TBox). Ce modèle a été élaboré selon la sémantique RDFS, qui était suffisante pour cette première application.

Ce modèle simple décrit essentiellement quelles sont les classes et leurs propriétés et quels sont les domaines (*rdfs:domain*) et codomaines (*rdfs:range*) de ces dernières. Les domaines des propriétés correspondent aux classes pouvant être caractérisées par cette propriété tandis que leurs codomaines renseignent sur la valeur de cette caractérisation. Une étiquette (*rdfs:label*) a été affublée à chacune de ces classes et propriétés, ce qui est requis pour le bon fonctionnement de l'application. Les étiquettes permettent de fournir des noms lisibles par les humains aux concepts ontologiques. Ces noms sont indépendants des URI qui désignent de façon unique ces concepts. Cela permet de facilement modifier l'étiquette et d'avoir plusieurs étiquettes pour un même concept possiblement en plusieurs langues. L'étiquette n'affecte aucunement les liens de ce concept avec les autres concepts ontologiques, découplant la signification du signifiant. L'utilisation obligatoire des étiquettes s'est avérée une constante dans les trois applications développées dans le cadre de cette recherche. Ces applications étant destinées à une utilisation humaine, les URI conviennent relativement mal à l'exploration des données par ceux-ci. Quoi que non explorées durant ces recherches, des techniques pour dériver automatiquement des étiquettes depuis des URI existent et pourraient être exploitées pour l'utilisation de modèles n'en contenant pas.

Finalement, ce modèle attribue aussi à chacune des propriétés des métadonnées de présentation qui seront utiles lors de l'utilisation des services CRUD par l'usager. Ces métadonnées servent principalement aux composantes génériques de l'application afin qu'elles puissent s'adapter aux données dont on ne connaît pas *a priori* le modèle. Dans les applications suivantes, ces métadonnées seront regroupées dans des ontologies de visualisation. Il y sera alors expliqué comment l'utilisation de telles ontologies est nécessaire au développement de composantes plus complexes, dans une perspective d'ingénierie dirigée par les ontologies.

3.2.3 Séparation conceptuelle des graphes

Le développement de l'application a posé le défi suivant : les données présentées à l'usager devaient refléter l'état le plus à jour de la base de données officielle de l'entreprise, tout en lui permettant d'ajouter et de modifier des individus des classes sans altérer le contenu provenant de la base officielle. De plus, l'usager devait pouvoir choisir de visualiser uniquement les données qu'il a personnellement enregistrées, uniquement celles de la base officielle, ou l'agrégation des deux.

La solution choisie pour résoudre ce problème a été de diviser le graphe RDF en plusieurs sous-graphes distincts (Figure 3.2). La première division sépare les données selon leur provenance, pour déterminer lesquels sont altérables par l'usager. La deuxième division les classe selon leur présentabilité, certaines données en nécessitant d'autres d'une provenance distincte pour être affichées.

Cette application a donc requis l'utilisation de plusieurs sous-graphes de données pour pouvoir faire la distinction entre les informations du domaine, les informations de présentation, la connaissance conceptuelle du modèle, les individus ainsi que les informations sujettes à l'édition et celles qui ne le sont pas. En outre, il était requis que ces informations soient intégrées de façon transparente par le UI. Pour ce faire, une agrégation virtuelle est réalisée par la BDT en fonction des sources de données que l'usager choisit de visualiser grâce à un menu des agrégations possibles.

L'ensemble des triplets a donc été séparé en six graphes sémantiques dans la BDT : l'ontologie de domaine (TBox), les informations de visualisation et quatre ABox.

Comme le montre la Figure 3.2, le ABox est séparé en quatre sous-graphes. La première séparation représente le fait que l'information peut provenir de deux sources distinctes : soit elle provient de la base de données officielle de l'entreprise, soit elle a été générée par des usagers. Pour des raisons de sécurité il est impossible pour les usagers de modifier l'information provenant de la base de données officielles. Ainsi, cette séparation permet de distinguer les informations qui sont modifiables par l'usager de celles qui ne le sont pas. En d'autres termes, les données issues de la BDR officielle sont classées dans des graphes de la BDT différents de ceux contenant des données créées par l'usager. De même, les données créées par les usagers, que ce soit avant ou pendant l'utilisation de l'application, ne seront jamais ajoutées à la BDR officielle.

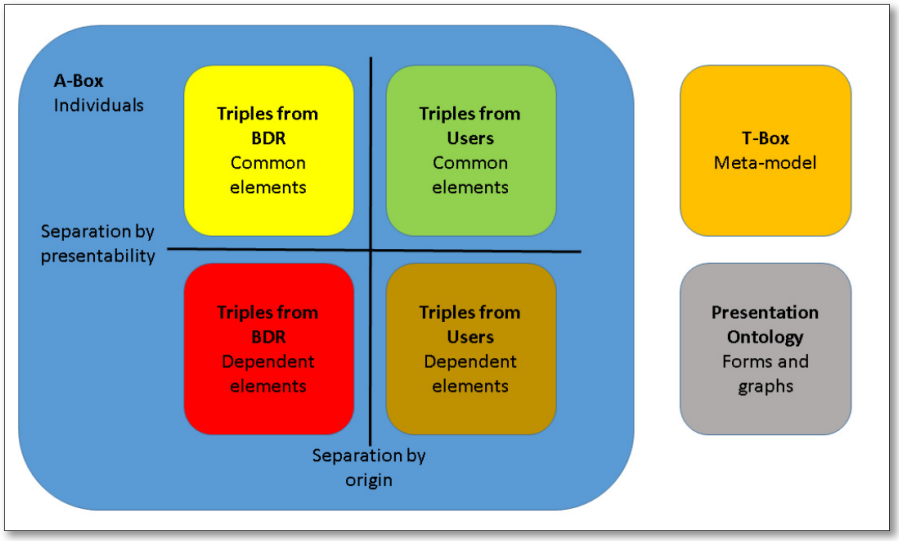

Figure 3.2 Organisation des graphes sémantiques
Source : Tiré de Bhérer, Vouligny, Gaha, & Desrosiers (2016)

Il est alors simple de synchroniser le graphe des informations non-modifiables avec la base de données officielles de l'entreprise. En effet, cette façon de faire assure de ne pas perdre les informations générées par les usagers lorsque les triplets provenant de la BDR officielle sont effacés pour être recréés. De plus, cette division permet la visualisation des deux jeux de

données indépendamment l'une de l'autre, mais ceci, uniquement si une autre séparation est faite.

Cette autre séparation isole les données dites dépendantes des données dites indépendantes. Ce second type de séparation, celui de la présentabilité, est plus complexe. Prenons le cas d'une mesure créée par un usager sur un transformateur qui lui est issu de la base de données officielle. En appliquant un filtre pour ne voir que les données créées par l'utilisateur, le transformateur est alors filtré et la mesure devient inaccessible. Une division relative à la dépendance des données vient permettre de visualiser exclusivement des mesures créées par les usagers, mêmes celles décrivant des transformateurs provenant de la base de données officielles.

La nécessité de ce type de séparation s'explique par le fait que dans cette première application, l'utilisateur doit choisir entre deux arbres pour explorer les données. À cette étape de la recherche, ces arbres ne sont pas auto-adaptatifs et leur structure est donc figée.

Ainsi, en reprenant l'exemple précédent, pour pouvoir visualiser une vue sur des mesures créées par un usager décrivant un transformateur provenant de la base officielle, il est alors nécessaire que la requête allant chercher les mesures ne discrimine pas les données de la classe des mesures et les données de la classe des transformateurs de la même manière.

C'est cet état des faits qui a entraîné la séparation des données d'une même provenance en deux sous-graphes : les éléments dits communs qui peuvent être visualisés par eux-mêmes et les éléments dépendants qui nécessitent des éléments communs pour être visualisés. Cette séparation est faite au niveau des classes, selon leur position dans le modèle et elle dépend essentiellement de la façon dont l'usager peut naviguer dans le TBox, soit, dans ce cas, par les différents arbres de navigation.

Le modèle de données et les métadonnées de présentation sont, quant à eux, isolés dans leur propre graphe, le premier pour pouvoir être agrégé de façon indépendante à n'importe quelle combinaison de ABox et le second pour pouvoir être réutilisé dans le développement d'autres applications. Le graphe de visualisation contient les types de champs de formulaires et les types de graphiques pouvant être générés par l'application. Il évoluera en ontologie de visualisation dans OM (voir section Ontologie de visualisation).

Lors de la récupération de données, une agrégation virtuelle des graphes désirés est faite dans *Oracle* selon la requête de l'usager. Cette façon d'arranger les données permet de maintenir les informations provenant de la base de données officielle à jour, par la destruction et la repopulation des deux graphes de cette provenance, tout en préservant la présentabilité des éléments qui dépendent d'éléments communs de provenances différentes. Lorsque l'usager utilise les services CRUD, la création d'individus se fait nécessairement dans les graphes locaux tandis que la modification ou la suppression de triplets n'est permise que si ces derniers ne proviennent pas de la BDR officielle. C'est une solution simple et efficace pour offrir les fonctionnalités de visualisation désirées tout en préservant la préséance de la véracité des données de la base de données officielle de l'entreprise.

Dans les applications subséquentes, les composantes deviendront de plus en plus auto-adaptatives, ce qui entraînera le besoin pour le chargement de toutes les classes d'une vue de manière indépendante. Cette façon de faire d'une part plus complexe pour la création des composantes et des requêtes, vient d'autre part simplifier le mécanisme présenté ici. Les filtres sur une classe étant alors appliqués indépendamment des autres classes, on peut donc obtenir le chargement de données provenant ou non de la base de données officielle dans une même vue sans avoir besoin de la séparation entre les classes dépendantes et les classes indépendantes.

3.2.4 Objet générique

Puisque le but de DATE consistait en la construction d'une application pouvant s'adapter à n'importe quel modèle de données, il a été nécessaire de concevoir une technique pour transporter l'information sur une classe, ses propriétés et ses individus de manière standards et générique depuis le serveur jusqu'au client. Cette technique a pris la forme d'un véhicule générique permettant au client de traiter les données provenant de n'importe quelle classe. Dans DATE la forme de ce véhicule était fixe, cependant des stratégies pour adapter automatiquement la forme du véhicule lors du développement de nouveaux composants Web seront présentées au CHAPITRE 6.

Une instance donnée de cet objet de transport d'information est bâtie grâce à des requêtes SPARQL génériques. La généralité de ce véhicule provient de l'utilisation des patrons de méta-modélisation du langage RDFS pour guider le choix de ses propriétés. Puisque le modèle ordonne les données selon des éléments de méta-modélisation (classes, propriétés, individus), il a été possible de concevoir un véhicule avec des propriétés fixes pouvant contenir n'importe quelle classe RDFS, ses propriétés et ses individus. Le véhicule contiendra donc des informations sur la classe (URI et étiquette), sur chacune de ses propriétés (URI, étiquette, codomaine et métadonnées de présentation) et sur chacun des individus liés à l'individu sélectionné dans l'arbre (URI, étiquettes et valeurs pour chaque propriété).

Le véhicule générique avait alors la forme présentée à la Figure 3.3. À la Figure 3.5 se trouve une version instanciée de ce véhicule.

- Accès :
  - Individu de la classe objet
  - Prédicat pont
- Classe :
  - URI de la classe
  - Étiquette (*rdfs:label*) de la classe
- Liste [Propriétés]
  - URI de la propriété
  - Étiquette (*rdfs:label*) de la propriété
  - Portée (*rdfs:range*) de la propriété
  - Présentation du champ de forme de la propriété
- Liste [Individus]
  - URI de l'individu
  - Étiquette (*rdfs:label*) de l'individu
  - Cartographies {Propriété: Valeur} (pour chaque propriété de la liste des propriétés)

Figure 3.3 Objet générique
Source : Adapté de Bhérer, Vouligny, Gaha, & Desrosiers (2016)

Il est à noter que chaque individu contient la cartographie de toutes les propriétés avec leur valeur pour cet individu particulier. C'est cette cartographie qui permet de remplir les grilles du UI puisque la liste des propriétés est utilisée pour construire les colonnes des grilles.

Quant aux accès, ils permettent de retracer le chemin qui a été parcouru dans la vue pour se rendre à cette classe (i.e. la propriété liant les individus des deux classes) ainsi que l'individu de la classe sujet utilisé pour sélectionner les individus de la classe objet. Le terme prédicat-pont sera utilisé pour indiquer la propriété liant la classe *sujet* à la classe *objet*. Se référer à la Figure 3.4 pour bien saisir le sens de ces termes. Les accès représentent en quelque sorte la raison pour laquelle la classe a été récupérée et ils sont plus facilement compris par un exemple. Dans l'application développée, on peut chercher toutes les classes ayant une propriété dont le codomaine est un individu de la classe Transformateur. La classe Transformateur est alors la classe *objet* et les classes trouvées sont les classes *sujets*. L'individu de la classe objet est donc le transformateur sélectionné par l'usager dans un arbre et le « prédicat-pont » est la propriété de la classe *sujet* reliant ses individus à des individus de la classe *objet*.

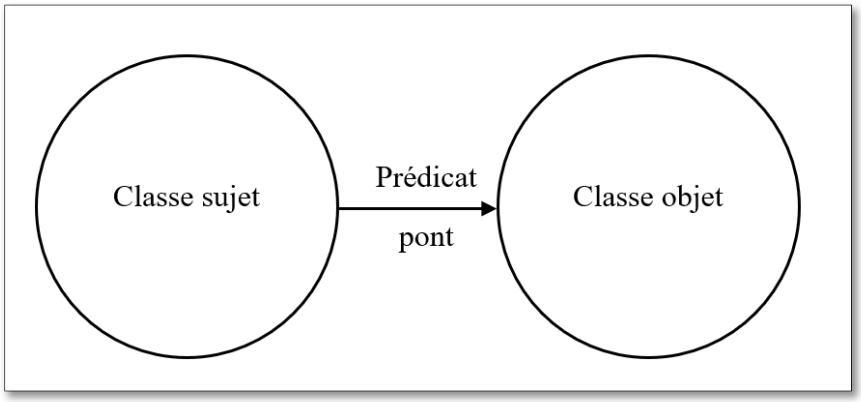

Figure 3.4 Prédicat pont

La Figure 3.5 présente un exemple d'instanciation de l'objet générique : l'utilisateur a sélectionné l'individu portant le numéro 123 (Accès –> ProvientDe). Le cadre interroge alors la BDT et trouve trois classes qui ont une relation d'association avec la classe des Transformateurs de puissance, soit la classe des Mesures, la classe des Interventions de maintenance et la classe des Associations de transformateurs aux postes électriques.

Ces trois classes sont alors recueillies une par une et chacune sera retournée au client dans un véhicule générique. L'exemple de la Figure 3.5 présente l'objet de la classe des Mesures. L'URI et l'étiquette de la classe sont retrouvés et ajoutés aux informations sur la classe. Les

propriétés de cette classe (dans la Figure 3.5 la concentration d'éthanol et la concentration de méthanol) sont ensuite colligées dans une liste où sont indiqués leur URI, étiquettes, codomaines et information de présentation. Par la suite, les individus de cette classe qui sont liés à l'individu 123 par la propriété de prédicat-point (Accès -> ParLaPropriété) sont colligés dans une seconde liste présentant la cartographie entre les propriétés de la liste précédente avec les valeurs spécifiques pour cet individu. Puis, les informations d'accès sont ajoutées à l'objet. Finalement, la liste des trois objets génériques est retournée au client.

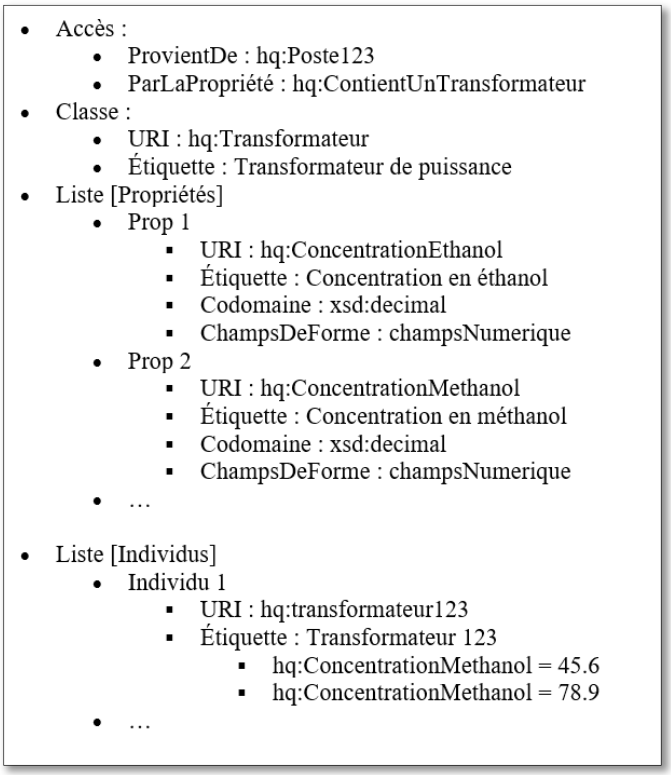

Figure 3.5 Exemple d'objet générique
Source : Adapté de Bhérer, Vouligny, Gaha, & Desrosiers (2016)

Dans cette application, chaque fois qu'un usager sélectionne une nouvelle feuille de l'arbre, le modèle ainsi que ses données sont interrogés. Ainsi, si on ajoute une nouvelle classe au modèle lié à la classe des feuilles, cette nouvelle classe sera présentée avec les autres dès la prochaine requête. De manière similaire, si les propriétés d'une classe sont modifiées, la nouvelle

mouture de la classe est présentée dès la requête suivante, sans que le code n'ait à être modifié. L'application est donc auto-adaptative à certaines modifications de son modèle de données.

Le véhicule générique qui vient d'être présenté peut prendre diverses formes. Dans cette recherche deux formes ont été étudiées : 1) grâce à un objet générique Java qui est transformé en une représentation JSON sur réception ou 2) grâce à un modèle générique de triplets sous le format JSON-LD.

Comme les formats JSON et JSON-LD partagent une structure de données similaires, ils peuvent tout deux être utilisés par la plupart des applications et des cadres de développement Web, avec la seule différence qu'un document en JSON-LD doit être traité par une librairie avant de pouvoir être utilisé. En effet, puisqu'un JSON-LD retient la structure en graphe des données qu'il contient, une librairie comme JSON-LD.js (JSON-LD, 2020) permet de le transformer en diverses représentations JSON par l'utilisation de cadres personnalisables par le développeur. En effet, cette librairie permet d'encadrer les graphes RDF dans un arbre de données non-redondant et de compacter les URI dans des mots-clés, produisant ainsi un JSON en bonne et due forme.

Utiliser un objet générique Java et le transmettre dans un format JSON est une façon naturelle de faire transiter l'information dans un paradigme orienté-objet (OO). Cependant, puisque la base de données provient du paradigme sémantique, garder les données sous format RDF est aussi judicieux.

Une importante différence entre les deux systèmes provient du fait que les requêtes génériques de forme SELECT dans le premier sont remplacées par des requêtes génériques de forme CONSTRUCT dans le second. L'utilisation de requêtes de type SELECT mène à une analyse (*parsing*) de l'ensemble des résultats dans un paradigme OO, i.e. l'objet générique Java. Les requêtes de type CONSTRUCT, d'un autre côté, mènent à la création d'un nouvel ensemble d'énoncés RDF. Par l'utilisation de la librairie Jena, on peut obtenir directement le résultat d'une requête SPARQL en format JSON-LD. En construisant des requêtes génériques CONSTRUCT, il est ainsi possible d'extraire les mêmes données d'une BDT que par l'utilisation des requêtes SELECT ayant mené au système avec objet générique Java. Ces requêtes sont conçues pour produire un graphe RDF ayant la même structure que celle de

l'objet Java, donc le client peut utiliser des transmissions provenant d'une méthode comme de l'autre, moyennant de légères modifications au code.

Peu importe la méthode utilisée, une fois la transmission complétée et les données envoyées du côté client, l'UI produit une matrice en deux dimensions pour chaque classe de la liste. Ces grilles montrent l'information en utilisant les étiquettes intelligibles pour les humains. L'utilisateur peut alors lancer des opérations CRUD sur les individus présents dans ces matrices à l'aide de formulaires.

3.2.5 Représentations visuelles

Cette application utilise du côté client des arbres extensibles pour présenter à l'usager un sous-graphe du graphe des données. L'usager peut choisir entre deux arbres. Le premier comporte deux niveaux : les branches sont les postes électriques contenus dans la base de données qui, une fois étendues, présentent leurs feuilles qui sont les transformateurs de puissance présents dans ces postes.

Le second arbre a trois niveaux : les premières branches sont des plages temporelles représentant les mois auxquels ont été prises les mesures. Elles s'ouvrent sur un deuxième niveau de branches représentant les postes électriques dont les transformateurs de puissance ont fait l'objet de mesures durant la période sélectionnée. Finalement, ces transformateurs sont les feuilles de cet arbre.

Lorsque l'usager choisit une feuille d'un des deux arbres (e.g. le transformateur 123), le client demande au serveur, grâce à une fonction générique, de récupérer dynamiquement toutes les classes *sujets* (e.g., Mesure) pointant vers la classe *objet* (e.g. Transformateur), c'est-à-dire, toutes les classes ayant une propriété dont le codomaine *(rdfs:range*) est la classe Transformateur. De chacune de ces classes seront tout d'abord dynamiquement récupérés toutes les propriétés, puis tous les individus pointant vers le transformateur sélectionné. Ces deux récupérations sont effectuées par la construction dynamique de requêtes SPARQL. L'utilisation de la propriété OPTIONAL permet de récupérer les individus et leurs valeurs, peu importe les propriétés instanciées pour chaque individu.

Pour chacune des classes *sujets* trouvées, le service enverra une réponse au client contenant l'objet générique construit à partir de ces requêtes. Grâce à ces informations, le client produit une grille en deux dimensions pour chacune des classes, qui peuvent être naviguées et triées par l'usager et grâce auxquelles il peut déclencher les opérations CRUD sur chaque individu séparément. Les opérations CRUD récupèrent l'objet générique, remplacent les valeurs à modifier, supprimer ou ajouter, puis retourne l'objet au serveur. Celui-ci traitera alors ces objets et ajoutera, modifiera ou supprimera les triplets de la BDT en conséquence.

Tel que mentionné, si des changements sont apportés au modèle par l'ajout ou la suppression de classes ou de leurs propriétés, le seul fait de cliquer sur la même feuille de l'arbre, permettra à l'usager de visualiser le nouvel état du modèle sans aucune modification des couches client ou serveur. Dans DATE, les nouvelles classes doivent absolument être liées à la classe des feuilles de l'arbre pour être visibles. Cela contraint la façon de créer le modèle et les possibilités de changements permissibles sur celui-ci pour que l'application fonctionne correctement. Cette contrainte signifie que pour permettre la récupération dynamique des classes *sujets* et de leurs individus, celles-ci doivent posséder une propriété les liant à la classe *objet*. Il a été nécessaire de créer des boutons permettant de récupérer directement et dynamiquement les propriétés et les individus des classes n'étant pas reliées à une classe des feuilles de l'arbre. Elles sont alors affichées grâce aux mêmes fonctions génériques par l'utilisation du même objet générique Java dans lequel les propriétés d'accès sont nulles. L'utilisation de l'objet générique permet alors l'utilisation des mêmes fonctions pour permettre la visualisation et la réalisation des opérations CRUD sur ces classes non liées.

3.2.6 Services CRUD

Les services CRUD sur les individus se font à partir de formulaires qui se créent dynamiquement en tirant profit des propriétés de la classe contenues dans le véhicule générique. Afin de permettre la vérification des données entrées par l'usager et pour diriger et faciliter cette saisie, un graphe RDF de propriétés de présentation a été créé, permettant de décrire comment le client devrait construire le champ de saisie de chaque propriété de domaine.

La création de chaque champ du formulaire se fait ainsi selon un type de présentation pour chacune des propriétés.

Tous les ajouts se font systématiquement dans un des deux modèles locaux, selon que la classe que l'on tente d'instancier soit considérée comme un élément commun ou comme un élément dépendant. Lors de la création du formulaire de modification ou de suppression, un service est appelé qui trouve la provenance des triplets décrivant l'individu pour interdire l'opération s'ils proviennent de la base officielle de l'entreprise. Les champs non modifiables sont alors grisés et désactivés. Un utilisateur voulant modifier un triplet de la base officielle devra passer par d'autres canaux de l'entreprise pour le faire. Il verra alors de telles modifications appliquées dans DATE au moment de la régénération de la BDT provenant de la BDR officielle de l'entreprise.

Cette application n'utilise que quatre types de champs : des champs numériques, des champs textes, des menus déroulants et des dates. Les menus déroulants appellent un service qui trouve la liste des modalités possibles d'une propriété et peuvent forcer ou non l'utilisateur à choisir parmi celles-ci. Même dans le cas où l'usager est forcé dans le choix des valeurs, l'application lui permet de ne mettre aucune valeur. Pour tous les champs permettant l'ajout, la modification ou la suppression de littéraux, les types des saisies correspondent aux codomaines (*rdfs:range*) des propriétés qui sont importées dans l'objet générique. Les cardinalités ont aussi un rôle à jouer dans le choix de ce type de champs.

3.2.7 Graphiques

Le graphe RDF de visualisation contient aussi des classes pour décrire des types de graphiques que l'usager peut vouloir visualiser, tels que des histogrammes, des nuages de point ou autres. Les propriétés contenues dans le modèle du domaine sont alors augmentées de triplets permettant d'indiquer leur utilisation dans la création d'un graphique. Lorsque ces propriétés sont présentes dans un objet générique, le client les détecte et crée une liste de graphiques que l'usager peut alors visualiser.

3.3 Conclusion

Dans ce chapitre a été présenté l'application DATE, développée pour un chercheur chimiste de l'IREQ afin de servir de tableau de bord pour faire la surveillance des transformateurs de puissance d'HQ. Grâce aux technologies du WS, DATE présente des rudiments d'auto-adaptabilité qui proviennent notamment de l'utilisation conjointe d'une ontologie de présentation, de requêtes génériques, d'un véhicule générique et de composantes Web génériques.

Grâce à D2RQ (D2RQ, 2020), une librairie permettant de cartographier une BDR dans le format RDF, un fichier de correspondance a été créé pour télécharger les données d'EVA dans la BDT de DATE. Celles-ci se conforment au modèle de données de DATE qui contient une dizaine de classes et une quarantaine de propriétés. Ces données sont traitées par différents calculs qui permettent entre autres de corriger les concentrations des molécules en fonction de la température de l'huile et de classifier les transformateurs selon leur état de santé.

DATE permet l'exploration ontologique à partir d'un arbre et les classes liées aux classes des feuilles de l'arbre sont retrouvées dynamiquement à chaque requête. Ainsi la simple modification de celles-ci dans l'ontologie entraîne la visualisation des changements sans n'apporter aucune modification au code et sans nécessiter de recompilation. Il en va de même pour les propriétés et les individus de ces classes.

Cette façon de faire et le besoin de séparer les données créées par l'utilisateur d'avec celles provenant de la base de données officielle a entrainé le besoin d'un mécanisme pour visualiser ces deux sortes de données séparément. Celui-ci a été implémenté grâce à une séparation conceptuelle du graphe des données en plusieurs ABox.

Les opérations CRUD sur les données s'effectuent par l'entremise de formulaires générées dynamiquement. Les types des champs des formulaires correspondent généralement aux codomaines des propriétés mais des propriétés de présentation peuvent aussi être utilisées pour diriger ce choix. Des propriétés de présentation servent également à diriger la construction de graphiques dans l'interface client.

Ce développement a été sélectionné comme banc d'essai pour démontrer les propriétés d'auto-adaptativité que peuvent apporter les bases de données sémantiques aux applications. Ce banc d'essai a tout d'abord servi à prouver que l'indépendance conceptuelle entre les modèles de données et les applications s'obtient de façon naturelle grâce aux technologies du WS. En effet, grâce aux standards de modélisation ontologique que sont RDF, RDFS et OWL, et au fait que les modèles de données sont enregistrés dans le même format que les données elles-mêmes, il a été possible d'écrire des requêtes génériques permettant à l'application de s'auto-adapter aux changements du modèle de données. Les changements permis sur les modèles pour que l'auto-adaptabilité fonctionne sont toutefois limités dans le cas de cette application.

La création d'applications auto-adaptatives repose en grande partie sur le fait que les composantes génériques du côté client puissent interpréter les données générées par les requêtes génériques du serveur. L'information puisée dans la base de données doit donc être mise dans un format générique compréhensible pour les composants de l'interface client. Ce format doit quant à lui pouvoir contenir tout type d'information provenant de la base de données. Lors de la réalisation de cette étape, une première itération de ce format a été développée et deux techniques de communication pour faire transiger les résultats des requêtes du serveur au client ont été comparées. La première technique prend la forme d'un objet JAVA générique qui est par la suite transformé par des librairies appropriées en format JSON pour être envoyé au client et utilisé par ses composantes. La deuxième laisse quant à elle l'information sous forme de graphes de triplets qui sont sérialisés en format JSON-LD, envoyés au client et transformés par celui-ci en JSON avant d'être utilisés par les composantes. La deuxième technique s'est avérée plus performante.

Au CHAPITRE 6, plusieurs sujets concernant DATE seront discutés. Cette application fait partie de la catégorie des navigateurs textuels et ses fonctionnalités seront comparées à d'autres outils de cette catégorie. Il s'agit d'une application destinée à des utilisateurs débutants et ses fonctionnalités seront donc comparées à ce qui est attendu d'une application conçue pour ce type d'utilisateur. La principale limitation de cette application, le fait que l'exploration ontologique y soit restreinte aux classes directement liées à la classe des feuilles de l'arbres, y sera abordée. Finalement, la principale leçon apprise et la principale contribution de ce chapitre

consistant en l'atteinte de l'indépendance conceptuelle par une judicieuse composition des requêtes en tablant sur les technologies du WS y sera présentée.

CHAPITRE 4

CONSTRUCTEUR DE REQUÊTES VISUELLES SPARQL D'HYDRO-QUÉBEC

La deuxième étape de cette recherche a consisté à développer un système de requêtes visuelles SPARQL. Contrairement à l'application précédente, le but est ici de construire une application dont le UI ne limite en rien l'utilisateur dans son exploration du modèle de données. Basé sur des constructeurs de requêtes SQL tels qu'Access ou Toad, cet outil en est l'équivalent sémantique.

Sa conception a été réalisée de façon à combler un manque dans la littérature et dans l'offre d'utilitaires sémantiques : transposer les enseignements des technologies relationnelles en ce qui a trait aux constructeurs de requêtes visuelles dans l'univers des technologies du WS afin de diminuer le temps d'apprentissage et la résistance au changement des utilisateurs face à ces nouvelles technologies.

Une analyse des fonctionnalités des outils relationnels et sémantiques existants permettra de démontrer qu'un tel constructeur de requêtes sémantiques peut contenir les mêmes fonctionnalités que ses pendants relationnels.

4.1 Contexte

Les éditeurs graphiques de requêtes, aussi appelé constructeurs de requêtes visuelles (CRV), sont une sous-catégorie des interfaces graphiques pour utilisateurs. Les BDR ont des CRV qui appuient les SGBD tels qu'Access (Access, 2020) ou Toad for Oracle (Toad for Oracle, 2020). Les BDT quant à elles, sont toujours à la recherche d'un éditeur de ce type qui soit intuitif, performant et versatile.

Comme mentionné précédemment, à l'IREQ, plusieurs études sur les applications des technologies sémantiques ont été menées, telles que Zinflou, *et al.* (2013) et Gaha, *et al.* (2013). Différentes bases de connaissances sémantiques ont donc été développées pour différentes applications. Afin de permettre aux ingénieurs d'HQ d'interroger ces bases de données sans

avoir à apprendre le langage SPARQL, il a été décidé de mettre sur pied un constructeur de requêtes visuelles SPARQL appelé le « Constructeur de requêtes visuelles SPARQL d'HQ » (CRV-HQ). Développé pour ressembler au CRV de Toad, un éditeur de requêtes visuelles SQL fortement utilisé dans l'entreprise, le CRV-HQ a été conçu pour permettre une utilisation intuitive à des usagers néophytes des technologies sémantiques, tout en les initiant au monde des ontologies. Cet outil permet de parcourir virtuellement n'importe quelle ontologie OWL, mais il a été développé autour de l'*International Electrotechnical Commission / Common Information Model* (IEC/CIM), l'ontologie des réseaux électriques (CIM, 2020).

4.2 Solution développée

À l'instar de l'application précédente, le CRV-HQ a été implémenté dans un environnement trois-tiers. Tel que son nom l'indique, il fait partie de la famille des constructeurs de requêtes visuelles. Très près du langage SPARQL, cet outil tente d'en simplifier l'utilisation tout en sacrifiant le moins possible de son expressivité.

Tel que mentionné précédemment un CRV doit supporter deux activités complémentaires mais opposées : l'exploration et la construction. Pour supporter ces activités, les paradigmes de visualisation qui ont été sélectionnés comme point de départ pour le développement du CRV-HQ sont la représentation en graphe, la représentation par formulaire et la représentation par facette.

4.2.1 Vue d'ensemble

L'utilisateur de l'application est tout d'abord confronté à un canevas vide bordé d'une barre des tâches et d'un arbre présentant toutes les classes de l'ontologie. La Figure 4.1, présente les divisions principales du UI et les fonctionnalités qu'elle offre à l'utilisateur.

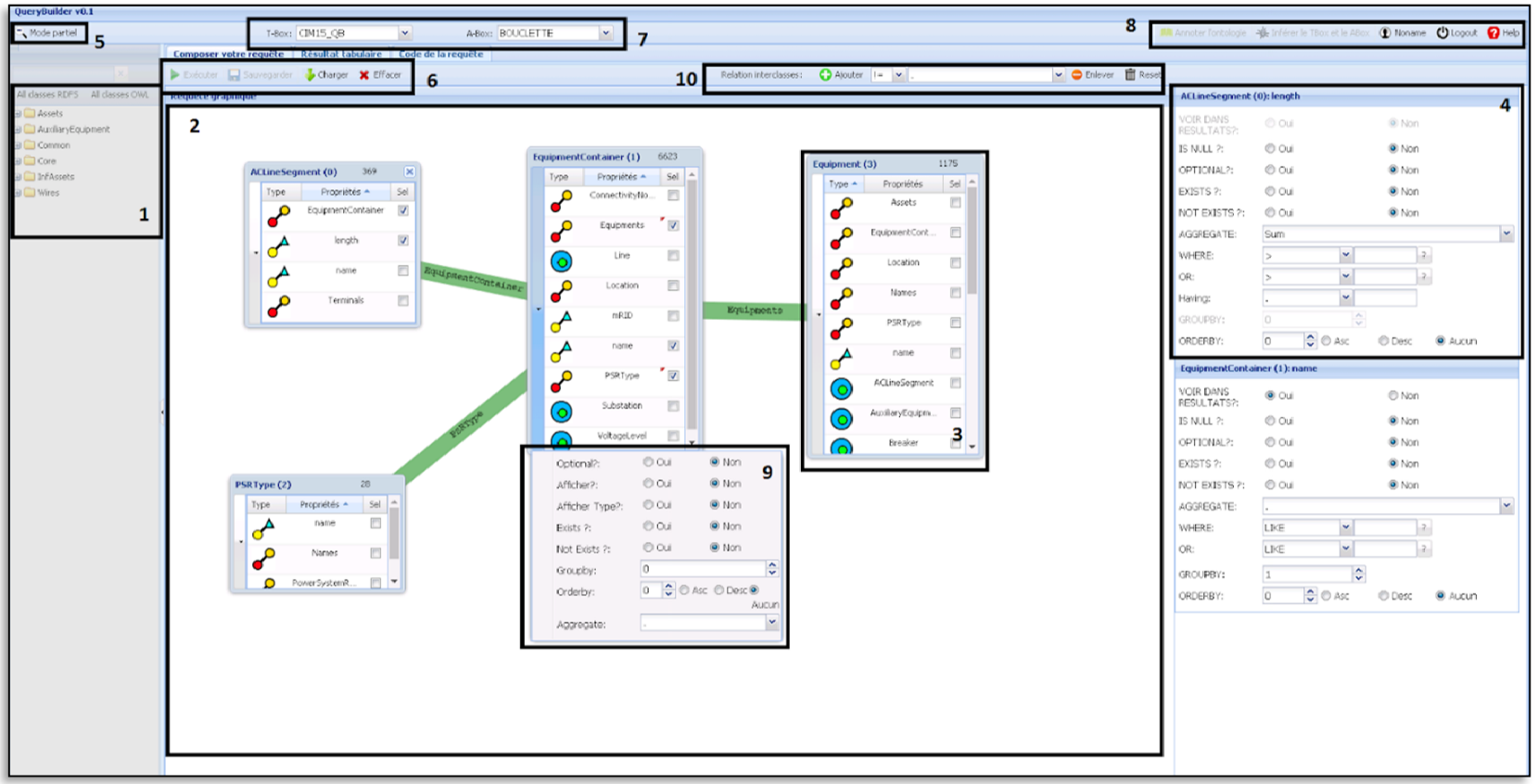

Figure 4.1 Vue d'ensemble du CRV-HQ

L'arbre des classes du modèle (1) permet de choisir le pivot de la requête. Le pivot d'une requête est simplement la première classe de celle-ci. On peut voir les individus de cette classe comme les sujets de la requête : tous les individus des autres classes qui seront retournés par la requête seront nécessairement liés de près ou de loin aux individus de la classe pivot.

Sur sélection de cette classe pivot, l'arbre est désactivé et l'exploration du modèle se poursuit dans l'espace graphique (2). Chaque classe est représentée par une boîte (3) montrant les propriétés de cette classe qui sont autant de facettes permettant de réduire l'espace de solution. La sélection d'une propriété de type objet, c'est-à-dire d'une propriété qui lie des individus de deux classes distinctes entre eux (icône ), entraîne l'ouverture d'une autre boîte représentant la classe du codomaine (*rdfs:range*) de cette propriété objet. Ainsi, l'arbre formé par les boîtes reliées entre elles forme le patron de la requête qui sera envoyée à la BDT.

La sélection d'une propriété de type donnée (icône ), c'est-à-dire d'une propriété dont le codomaine est un type de littéral (e.g. *xsd:string, xsd:float*, etc), entraîne l'ouverture d'un formulaire de filtre (4) où une sélection de fonctionnalités du langage SPARQL permet de filtrer l'espace de solution selon cette propriété.

Finalement, la sélection d'une propriété de subsomption (icône ) entraîne la transformation de la boîte en la boîte de sa sous-classe. Les propriétés présentées dans la boîte sont alors modifiées en conséquence.

Ce sont les trois mécanismes permettant l'exploration du modèle, la construction visuelle de la requête et la réduction de l'espace de solution. Plusieurs autres fonctionnalités viennent soutenir ces mécanismes, soit en tirant parti des sémantiques du langage SPARQL, soit en implémentant des paradigmes de représentation visuelle et d'aide à l'exploration. Parmi celles-ci, on retrouve les suivantes :

- Un bouton permet de n'afficher que les classes et propriétés instanciées du modèle dans l'arbre et dans les boîtes, ou encore d'en présenter toute la collection (5);
- Le nombre d'individus d'une classe est affiché en haut à droite de la boîte correspondante;
- Les requêtes créées par l'outil peuvent être sauvegardées et chargées (6);
- On peut utiliser l'outil avec n'importe quel modèle OWL qui implémente certaines sémantiques de base (7);

- L'inférence entre le jeu de données et le modèle, l'annotation du modèle et la gestion de la session utilisateur sont faites à partir des boutons supérieurs droits de la barre des tâches (8);
- Des fonctionnalités SPARQL peuvent être déclarées au niveau des classes (9);
- Des fonctionnalités de comparaison de classes peuvent être utilisées à partir de la barre des tâches de l'onglet de la requête visuelle (10);
- Des fonctionnalités d'exemples et de statistiques sur les littéraux sont disponibles à partir des filtres;
- Les boîtes peuvent être réorganisées dans le canevas graphique et être redimensionnées à la guise de l'utilisateur.

Lorsque l'usager a terminé la construction de sa requête, il peut ensuite l'exécuter (6). Le client compilera alors les différentes composantes de celles-ci dans un graph JSON-LD qui sera envoyé au serveur. Un jeu de fonctions récursives traite ensuite ce graphique pour composer une requête SPARQL de type SELECT qui est envoyée à la BDT. La requête et son résultat sous forme de matrice en deux dimensions sont finalement retournés au client Web pour être affichés.

Des fonctionnalités qui ne sont pas visibles sur la Figure 4.1 sont alors mises à la disposition de l'usager :

- La possibilité de trier les résultats;
- La possibilité de changer l'ordre des colonnes présentées dans les résultats;
- La possibilité d'exporter la grille des résultats sous forme de fichier CSV (en anglais *Comma-Separated Values*);
- La possibilité de voir la requête SPARQL textuelle fabriquée à partir du modèle visuel;
- La possibilité de modifier manuellement cette requête textuelle.

Cet outil, son fonctionnement et ses spécificités de conception seront maintenant présentés.

4.2.2 Préparation de l'environnement de travail

L'usager qui souhaite utiliser le CRV-HQ doit tout d'abord préparer son environnement de travail. Il doit premièrement choisir l'ontologie qu'il veut explorer. L'ontologie est le graphe du modèle des données (TBox). Un menu déroulant des TBox, permet à l'usager de choisir un graphe RDF parmi ceux disponibles dans son compte Oracle.

Suivant la sélection, l'application vérifie automatiquement si ce TBox est déjà annoté et, si ce n'est pas le cas, invite l'usager à le faire en appuyant sur le bouton correspondant. L'annotation du modèle sert à mimer le principe d'héritage des propriétés des classes du paradigme orienté-objet (OO). L'enjeu est ici de réconcilier ces deux paradigmes qui se chevauchent.

Cette réconciliation des paradigmes OO et de la logique de premier ordre est un enjeu qui fera l'objet d'une discussion à la section Paradigmes de classification et d'héritage du CHAPITRE 5. Pour l'instant, il est suffisant de savoir que, dans le CRV-HQ, cet enjeu a été abordé par l'ajout d'annotations dans le modèle de données qui permettent d'avoir une représentation OO de l'héritage des propriétés.

Une fois le modèle annoté, l'utilisateur doit sélectionner un graphe de données (ABox). La requête implicitement créée par l'usager sera effectuée sur une agrégation virtuelle du TBox et de l'ABox. Ce type d'agrégation virtuelle est commun dans l'utilisation des technologies du WS. Le modèle de données est typiquement séparé des jeux de données et l'agrégation est faite selon le ou les jeux de données que l'on veut interroger.

4.2.3 Inférences

Une fois l'agrégation réalisée, l'inférence logique sur les données est alors possible. L'engin d'inférence utilise des règles logiques pour déduire les informations implicites se trouvant dans les données en se basant sur les sémantiques RDF, RDFS et OWL contenues dans le modèle de données. L'inférence est au cœur de la présente recherche; elle a fortement orienté les choix de conception du CRV-HQ et de l'application subséquente.

Cette inférence a un coût computationnel considérable. Afin d'augmenter la performance des requêtes, Oracle préconise l'approche du chaînage avant. Cette inférence est effectuée *a priori* sur l'ensemble du jeu de données, avant que les requêtes ne soient lancées.

Sur sélection du TBox et de l'ABox par l'utilisateur, l'application détecte donc automatiquement si ceux-ci ont déjà été inférés ensemble. Si ce n'est pas le cas, l'usager sera invité à le faire en utilisant le bouton approprié.

L'inférence peut venir en plusieurs saveurs, correspondant à des sous-ensembles des jeux de règles RDFS et OWL. OWL peut être vu comme le jeu de règles ultime permettant d'harnacher l'expressivité complète de la logique de premier ordre. Cela peut rendre le processus d'inférence indécidable en plus d'être coûteux à effectuer. C'est pourquoi plusieurs variantes d'OWL ont été proposées dans la littérature afin d'adapter l'inférence aux formats des ontologies. Par exemple, la saveur OWL 2 EL, sera préconisée en présence d'une ontologie présentant un nombre élevé de classes alors qu'une autre saveur, OWL 2 QL sera préconisée pour les ontologies contenant peu de classes largement populées. Il est aussi possible de sélectionner manuellement les règles à effectuer, ce qui a été fait ici.

Dans le CRV-HQ, l'inférence effectuée contient sept règles. Tout d'abord les patrons d'inférence 9 et 11 de RDFS qui infèrent les relations de subsomption. Ensuite, une règle OWL pour inférer les patrons générés par l'axiome *owl:inverseOf* permet de spécifier les relations inverses.

Puis, quatre règles maisons servent à annoter les propriétés *rdfs:subClassOf*, *owl:ObjectProperty* et *owl:DatatypeProperty* ainsi que les classes ayant au moins un individu. Ces annotations permettent d'afficher à l'usager uniquement les entités ontologiques (classes et propriétés) qui sont instanciées, c'est-à-dire qui sont utilisées dans le ABox. En effet, le TBox contiendra toutes les classes et toutes les propriétés, mais les différents jeux de données ne feront souvent usage que d'un sous-ensemble de celles-ci.

L'usager pourra ainsi faire le choix entre une exploration du modèle complet ou seulement du sous-ensemble de ses sémantiques utilisées dans un jeu de données en basculant du mode complet au mode partiel. Ce mécanisme implique que l'arbre des classes présentera une

arborescence différente selon que l'usager soit en mode complet ou en mode partiel, tel qu'on peut le voir dans la Figure 4.2.

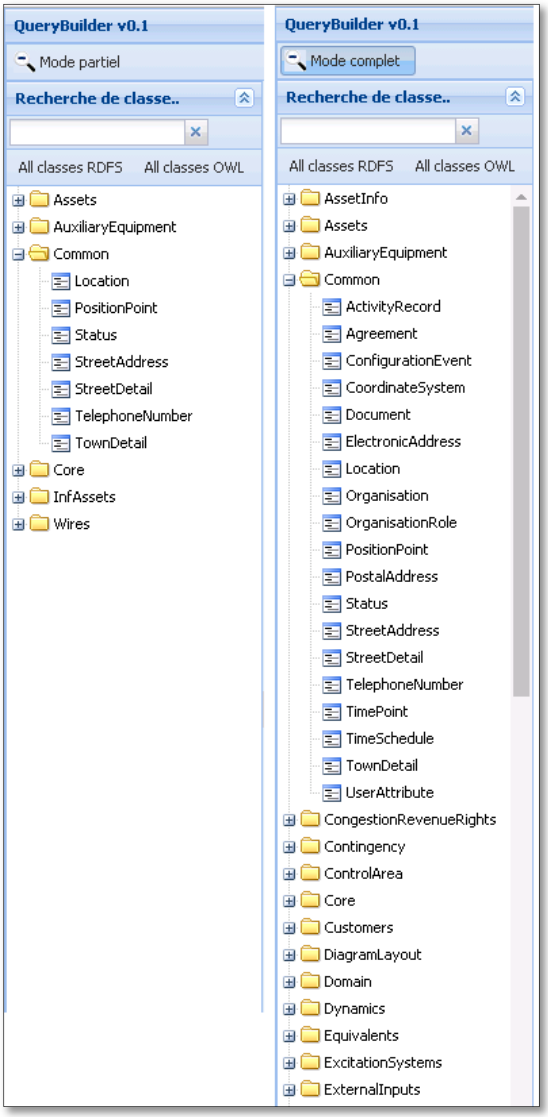

Figure 4.2 Arbre partiel et arbre complet

4.2.4 Exploration ontologique

Dans le CRV-HQ, la première phase de l'exploration ontologique consiste donc à sélectionner une classe qui deviendra le point de départ de la requête, c'est-à-dire la classe pivot, grâce à

l'arbre des classes. La deuxième phase se déroule quant à elle dans l'espace de canevas graphique (voir Figure 4.3).

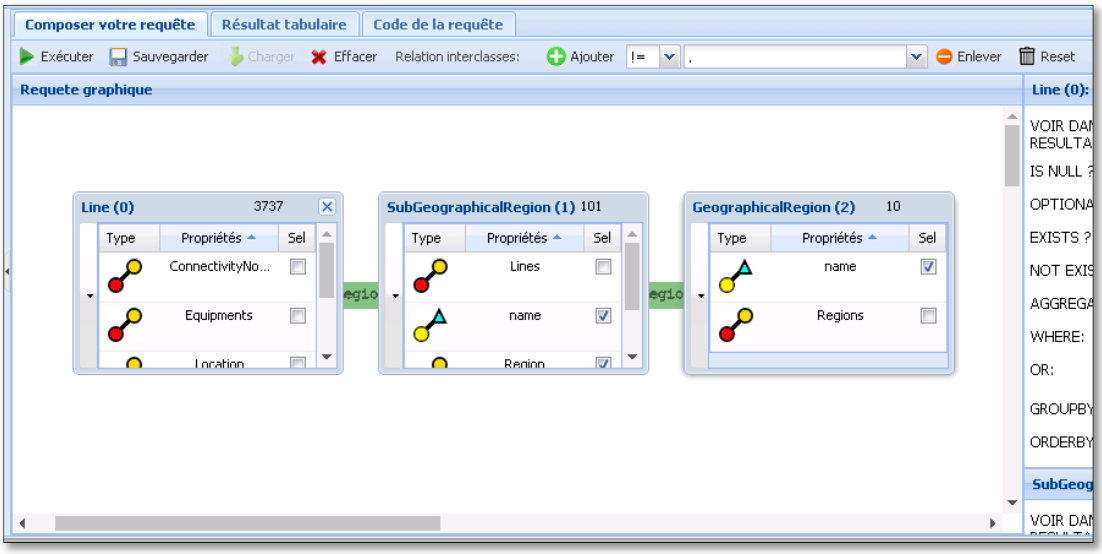

Figure 4.3 Canevas graphique

Une fois la classe pivot sélectionnée, une boîte représentant cette classe (Figure 4.4) est ajoutée dans l'espace de travail.

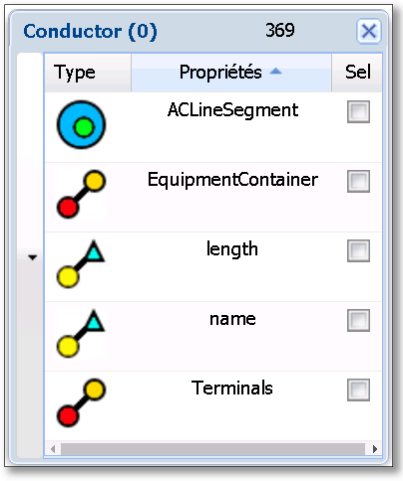

Figure 4.4 Petite boîte

Dans la boîte de la classe sont contenues toutes les propriétés de cette classe et de ses superclasses. Tel que mentionné précédemment, pour imiter l'héritage du paradigme OO, le modèle a dû être annoté avant sa première utilisation, autrement la boîte présenterait les propriétés de la classe et de ses sous-classes. Lors de ce processus, la relation *CRVHQ:directProp* qui relie chaque classe à toutes ses propriétés et aux propriétés de ses superclasses est ajoutée au modèle. Il a été choisi d'annoter le modèle pour augmenter la performance des requêtes lors de la population des boîtes. En effet, des requêtes auraient pu être effectuées à chaud pour déterminer ces propriétés, mais la performance en est alors grandement diminuée. Ce choix de conception est discuté plus en détail au CHAPITRE 5.

Trois types de propriétés sont présentés dans les boîtes : les relations de subsomption (*rdfs:subClassOf*), les relations d'association (*owl:ObjectProperty*) et les relations de type données (*owl:DatatypeProperty*). La deuxième phase d'exploration du modèle se fait à partir des relations d'association qui sont représentées par l'icône . La sélection d'une telle propriété provoque l'ouverture d'une nouvelle boîte représentant la classe codomaine (*rdfs:range*) de la relation (Voir Figure 4.5). Les deux boîtes sont reliées par une ligne présentant l'étiquette (*rdfs:label*) de cette propriété. Décocher cette propriété entraîne la fermeture de la boîte correspondante et de toutes les boîtes associées.

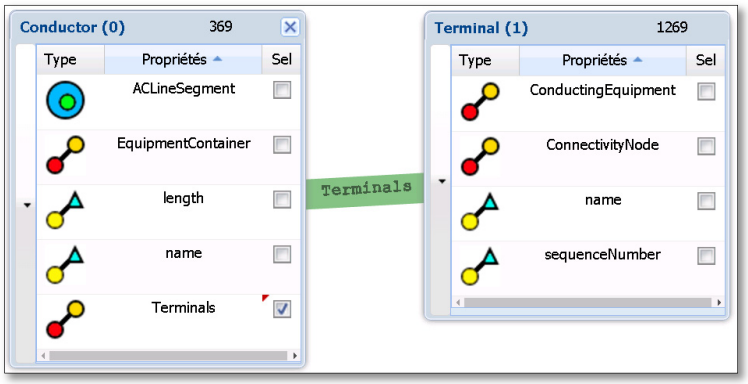

Figure 4.5 Deux petites boîtes

Les relations de type données sont représentées par l'icône . La sélection d'une telle propriété provoque l'ouverture d'un formulaire de filtre (voir section Espace des formulaires

de filtres de l'ANNEXE I). Ces filtres offrent plusieurs options afin de restreindre l'espace des résultats (voir Figure 4.6).

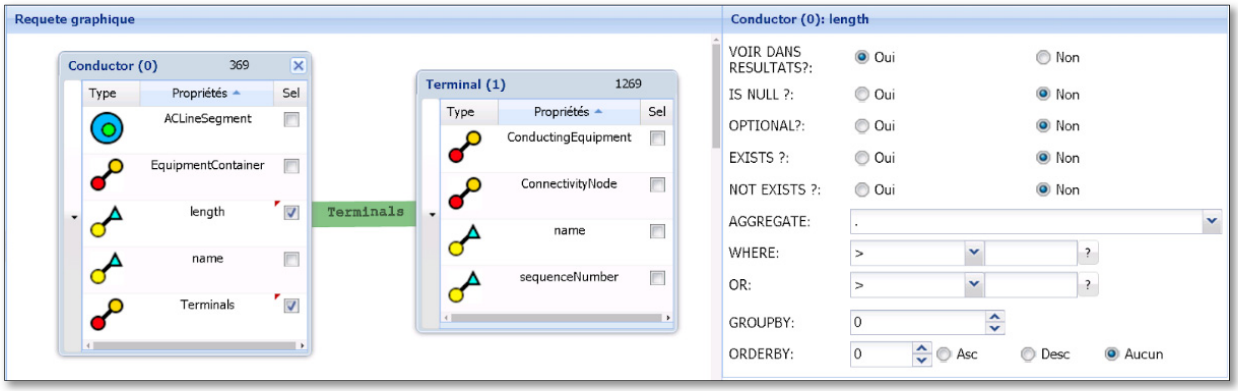

Figure 4.6 Deux petites boîtes et un filtre

Les relations de subsomptions sont représentées par l'icône . Cocher une telle propriété provoque la transformation de la classe en la sous-classe visée par la propriété de subsomption. Dans la Figure 4.7, on voit le même exemple qu'à la Figure 4.5, après que la propriété *ACLineSegment* ait été sélectionnée.

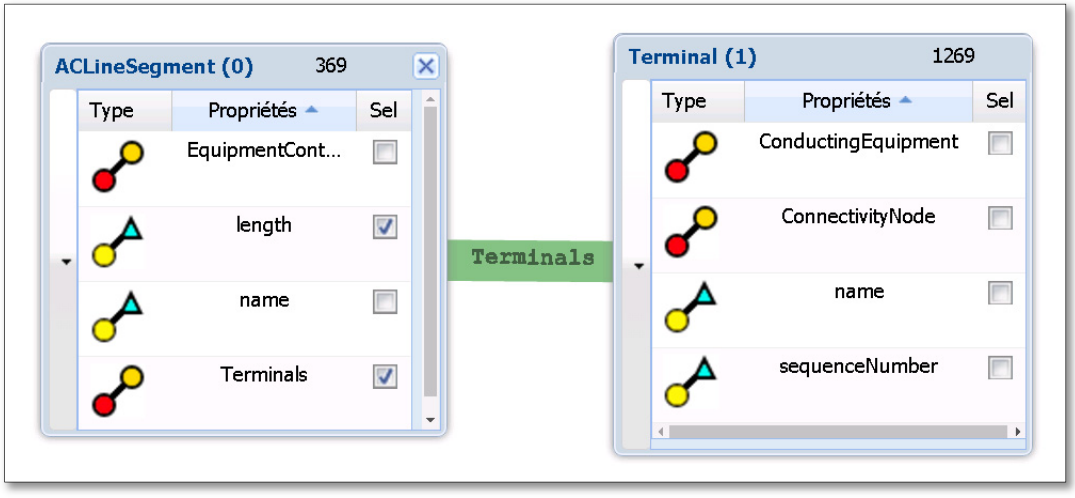

Figure 4.7 Spécialisation d'une classe
La classe *Conductor* est devenue sa sous-classe *ACLineSegment*

La classe *Conductor* est alors devenue sa sous-classe *ACLineSegment*. Puisque le nombre d'individus est resté 369 dans les deux cas, on peut déduire que tous les conducteurs présents dans ce jeu de données sont des segments de lignes à courant alternatif. Puisque dans le paradigme OO, une sous-classe hérite des propriétés de sa superclasse, aucune des autres propriétés déjà sélectionnées ne disparaît, ce qui assure que tous les éléments déjà ajoutés à la requête par l'utilisateur restent valides lorsque la classe se spécialise. En revanche, de nouvelles propriétés ont pu être ajoutées à la classe et celles-ci sont désormais visibles et activables dans la boîte. La relation inverse, de généraliser une classe en sa superclasse, n'a pas été implémentée.

À la gauche des boîtes représentant les classes, un bouton permet d'ouvrir un menu contextuel permettant de spécifier des clauses SPARQL sur la classe (voir Figure 4.8). Ces fonctionnalités et plusieurs autres sont offertes à l'usager pour préciser sa requête. Elles sont présentées à l'ANNEXE I.

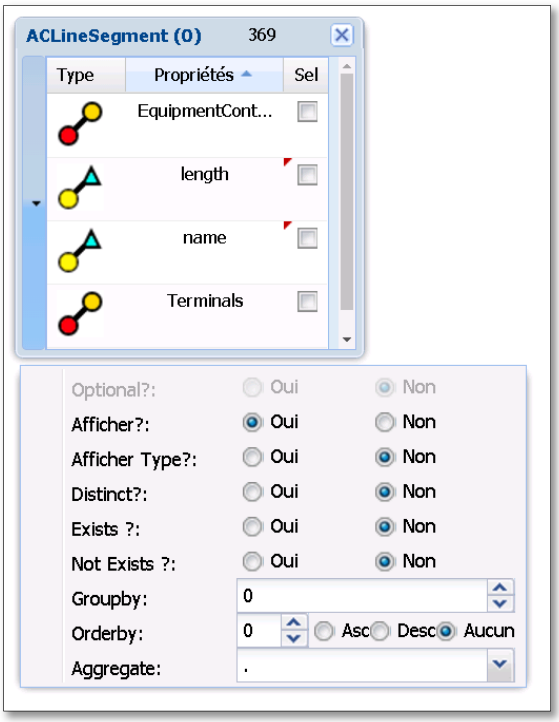

Figure 4.8 Menu de la petite boîte

Une fois sa requête construite, l'usager peut l'exécuter. Le serveur transforme alors le graphe de la requête graphique en une requête SPARQL et transmet celle-ci à la BDT pour qu'elle soit effectuée. La requête SPARQL et les résultats sont retournés au serveur qui les transfert à l'interface client. Celle-ci les affiche dans des onglets de l'espace de travail.

La Figure 4.9 montre l'onglet des résultats d'une requête présenté sous forme de grille (celle-ci ne comporte qu'une seule ligne).

Figure 4.9 Tableau des résultats

Avec les résultats est aussi ramenée la requête SPARQL textuelle. L'utilisateur peut la visualiser grâce au troisième onglet de l'espace de travail. Les usagers expérimentés peuvent modifier manuellement la requête et exécuter la version modifiée. Les résultats sont alors affichés dans l'onglet des résultats, mais la requête graphique n'est cependant pas modifiée en conséquence.

De plus amples renseignements sur l'interface graphique du CRV-HQ, ses fonctionnalités et leur implémentation sont présentées à l'ANNEXE I.

4.2.5 Communication entre l'interface et le serveur

Une fois sa requête construite, l'usager peut l'exécuter et la sauvegarder. Dans les deux cas, la requête visuellement construite sera tout d'abord sérialisée dans un graphe RDF par la partie cliente du CRV-HQ. C'est ce graphe qui sera envoyé du côté serveur grâce au format JSON-LD pour être traduit en SPARQL ou être sauvegardé dans la base de données des requêtes. Sérialiser les requêtes visuelles en format RDF a plusieurs avantages. Le premier provient du

fait que les requêtes peuvent être sauvegardées dans une BDT et ne nécessitent donc pas l'utilisation d'une BDR complémentaire. Le deuxième est que l'agrégation de l'ontologie et du graphe des requêtes permet d'analyser facilement quelles parties de l'ontologie sont les plus observées et comment les utilisateurs exploitent celles-ci. Finalement, la requête visuelle prenant la forme d'un arbre, des fonctions récursives permettent de facilement la sérialiser en RDF ainsi que de transformer ce RDF en SPARQL.

Les arbres, la plupart des menus déroulants de l'interface client et les listes des propriétés contenues dans les boîtes sont populés par des requêtes CONSTRUCT. Ce type de requête permet à la fois une plus grande complexité que les requêtes SELECT, d'obtenir un résultat sous forme de graphe et de diminuer les appels au serveur. Les échanges entre le client et le serveur se font pour la plupart en JSON-LD. Lors de la transformation de la requête visuelle en graphe, les composantes du cadre de développement Web sont transformées automatiquement en JSON par celui-ci. La librairie JSON-LD.js (JSON-LD, 2020) est alors utilisée pour transformer ce JSON en JSON-LD.

Les résultats de la requête, quant à eux, sont envoyés depuis le serveur vers le client dans le format JSON. Les requêtes visuelles effectuées par l'usager étant des requêtes de type SELECT, l'ensemble des résultats prend la forme d'une matrice en deux dimensions. JENA offre la possibilité d'écrire les résultats des requêtes SELECT SPARQL directement dans le format JSON et les librairies Javascript peuvent donc les utiliser tels quels. Il en va de même pour trouver les listes des modalités des propriétés.

Finalement, pour les requêtes plus simples sur la BDT, comme pour savoir si un graphe est annoté ou si le TBox et le ABox sont déjà inférés, des requêtes de type ASK sont utilisées.

4.2.6 Partie serveur

Le serveur est programmé en Java et comprend toutes les fonctions essentielles du CRV-HQ. Au cœur du serveur se trouve l'engin permettant de transformer la requête visuelle en requête SPARQL. La forme choisie pour les requêtes, soit l'arbre avec une racine unique, a permis l'implémentation rapide et efficace de cet engin de traduction.

Pour effectuer cette traduction, celui-ci traite les nœuds de la requête un par un, en commençant par la racine unique et en parcourant systématiquement toutes les branches grâce à des fonctions récursives. La forme RDF de la requête graphique a été conçue pour simplifier son parcours de façon itérative.

La requête finale est constituée de trois parties : sa tête, son corps et ses pieds. La tête correspond à liste des propriétés à sélectionner et toutes les opérations appliquées sur celle-ci qui se retrouvent entre les clauses SELECT et WHERE. Le corps de la requête contient les patrons de triplets ainsi que les fonctions de tri qui permettront de filtrer les résultats. Finalement, les pieds correspondent aux arguments supplémentaires qui viennent modifier la requête comme ceux d'agrégation, de pagination ou de classement des résultats. À chaque nœud, l'engin ajoute des éléments à chacune des trois parties. Une fois la requête complètement parcourue, les trois parties sont réunies et la requête est exécutée.

4.3 Conclusion

Dans ce chapitre, le CRV-HQ a été présenté. Celui-ci permet d'explorer n'importe quelle ontologie OWL avec différents jeux de données et ces TBox et ABox se sélectionnent par l'UI. Des inférences et des annotations sont effectuées sur l'agrégation virtuelle du TBox et du ABox *a priori* afin d'augmenter les performances de l'outil.

L'exploration ontologique se fait tout d'abord grâce à un arbre présentant les classes du TBox. Elle se poursuit dans un espace graphique où les classes sont représentées par des boîtes contenant les propriétés de celles-ci. Les propriétés d'association permettent à la fois d'explorer le modèle et de construire visuellement une requête. Les propriétés de type données permettent de restreindre l'espace des résultats. Les propriétés de subsomption permettent de spécialiser les classes de la requête.

Les requêtes sont exécutées et sauvegardées dans des graphes RDF sérialisés dans des JSON-LD issus de la transformation automatique des composantes Web en JSON. Le serveur transforme ces graphes RDF grâce à des fonctions récursives en des requêtes SPARQL de type

SELECT. Les résultats sont finalement ramenés en JSON et directement présentés dans l'onglet des résultats.

Le CRV-HQ marque le début de l'utilisation de l'inférence dans les applications de cette recherche. Cependant, il est loin de prendre en considération toute la richesse expressive de RDFS et OWL. Il est évident que l'outil gagnerait à mettre à profit plus des possibilités sémantiques de ces langages. Cela dit, les expériences faites qui seront discutées plus loin concluent que cette richesse vient avec un coût de calcul et d'implémentation rendant son utilisation prohibitive dans le cadre d'applications où le temps de réponse doit se rapprocher le plus possible de l'instantanéité.

Vue la nature de recherche de ce développement, aucun cahier de charge officiel n'a été produit, sinon des cahiers de charges informels. C'est pourquoi un tel document n'a pu être présenté ici.

Au CHAPITRE 6, plusieurs sujets concernant le CRV-HQ seront discutés. Ses fonctionnalités seront comparées à celles des outils semblables sémantiques mais aussi à celle d'un outil relationnel. Cette dernière comparaison permettra de confirmer l'hypothèse qu'on peut transposer les fonctionnalités déjà connues et appréciées des CRV relationnels vers les CRV sémantiques. Les enjeux de l'exploration ontologique, de l'optimisation des requêtes et de la pagination y seront exposés. Y seront aussi abordés les réflexions théoriques sur l'acceptabilité d'un tel outil, sa généricité, son expressivité et sur la réduction de la charge cognitive qu'il peut entraîner.

De ces discussions, on conclura que l'outil reste complexe pour les utilisateurs débutants, que les résultats ne sont pas présentés au fur et à mesure que l'usager compose une requête et que des fonctionnalités issues de l'utilisation des technologies du WS difficilement réalisable dans le monde relationnel sont souhaitables. Ces constats ont fortement influencé la conception de l'application du chapitre suivant. On y verra que leur implémentation est loin d'être triviale.

CHAPITRE 5

OWL MONKEY

L'étape 3 a consisté à concevoir et développer une méthodologie de génération d'applications sémantiques auto-adaptatives grâce aux connaissances et techniques explorées dans les deux outils présentés aux étapes précédentes. Cette méthodologie est implémentée dans un cadre de développement qui a été utilisé pour construire une première application dont la mise en production chez HQ s'est finalisée en décembre 2019.

Cet outil s'appelle Owl Monkey (OM) et peut être considéré comme un hybride entre un navigateur sémantique et un constructeur de requêtes visuelles permettant de construire des vues sur les données de façon itérative. Puisque ce type d'outil n'a pas été observé dans la littérature, la liberté a été prise de déclarer qu'il s'agissait d'un constructeur de vues. Il représente un compromis entre les deux premières applications présentées dans cette recherche : il offre à la fois beaucoup plus d'expressivité que DATE et beaucoup plus d'utilisabilité que le CRV-HQ.

L'intérêt d'OM repose donc d'une part sur son utilisabilité : l'utilisateur composant sa vue grâce à des mécanismes simples en voyant constamment les données qui en résultent, il peut aisément arriver au résultat souhaité même avec une connaissance quasi-inexistante du langage de requête et du modèle de données.

L'expressivité est son autre source d'intérêt : l'outil présente à l'utilisateur des fonctionnalités lui permettant de construire un très grand nombre de vues et quoi que toutes les sémantiques disponibles dans le langage de requête ne soient pas encore implémentées, l'approche permettra de le faire éventuellement au travers de composantes protégeant les néophytes de la complexité de SPARQL.

Finalement, sa généricité et son auto-adaptivité permettent à cet outil de fonctionner avec n'importe quelle ontologie OWL et entraînent que l'évolution des connaissances à l'intérieur d'une ontologie n'entraîne pas de réécriture du code ou de recompilation de l'application.

Owl Monkey tire son nom du douroucouli commun (*Aotus trivirgatus*), un singe de nuit des forêts tropicales d'Amérique du Sud dont le visage blanc et le style de vie nocturne lui ont valu le surnom de singe-hibou. On dit de ce nyctalope qu'il est d'une agilité exceptionnelle pour se maintenir en équilibre dans les arbres et sauter de branche en branche. Il chasse en outre en se déplaçant très rapidement, sans faire le moindre bruit grâce à sa légèreté et sa souplesse. Il était donc un totem parfait pour ce cadre de développement puisqu'il est particulièrement bien outillé pour parcourir les arbres de la connaissance qui sont écrits en OWL et qui sont baignés dans la noirceur constante de leur propre ramage.

5.1 Contexte

OM est un outil d'exploration ontologique destiné à des usagers néophytes des technologies sémantiques. Il a comme objectif de permettre la création de vues utiles dans un contexte d'entreprise et immédiatement utilisables pour échanger de l'information entre diverses parties prenantes. Il peut en outre être utilisé pour modifier les données des bases de connaissances sous-jacentes.

La première application construite grâce à OM s'appelle PPAL : Performance des paliers autolubrifiants. Cette application sert à enregistrer et partager les données de recherche produites par des chercheurs en science des matériaux de l'IREQ. Les ingénieurs de centrales hydro-électriques de l'entreprise peuvent ensuite utiliser PPAL pour déterminer les meilleurs matériaux entrant dans la conception d'installations hydroélectriques grâce aux diverses fonctionnalités d'OM. De plus, les employés s'occupant de la mise en service, de l'entretien et du retrait des pièces sont appelés à utiliser l'outil pour enregistrer des retours d'expérience sur l'utilisation de ces matériaux. Ceci a pour objectif de bonifier la base de connaissances en jumelant l'expérience pratique avec les données théoriques. Cela permettra donc à long terme de constituer une base de connaissance d'expériences terrain d'utilisation de ces matériaux qui viendra complémenter les données de recherche.

Étant donné la nature des utilisateurs (néophytes des technologies informatiques et *a fortiori* des technologies sémantiques), l'interface de PPAL (et d'OM) a été conçue pour être simple d'utilisation et rapide à maîtriser.

La construction d'une application grâce à OM se fait en deux étapes : la création du modèle de données, c'est-à-dire de l'ontologie, et la création des vues grâce au mode complet pour super-utilisateur d'OM. Le mode simple d'OM permet ensuite de protéger les utilisateurs normaux en ne leur donnant accès qu'à un sous-ensemble des fonctionnalités d'OM. Les utilisateurs normaux n'ont donc accès qu'à un navigateur sémantique, alors que les super-utilisateurs ont entre les mains un constructeur de vues.

Dans le cas de PPAL, la construction du modèle s'est faite à partir de données de recherche déjà existantes, des prévisions des données de recherche à venir ainsi que des besoins pour pouvoir ajouter les données de retours d'expérience. L'ontologie contient une trentaine de classes dont cinq d'énumérations, une quarantaine de propriétés objet, environ 80 propriétés de type données et une quinzaine de propriétés de subsomption.

À la suite de la création d'une version préliminaire du modèle de données, des rencontres avec les parties prenantes ont permis d'ajuster la nomenclature et la structure du modèle. La nature auto-adaptive d'OM permet de faire de tels changements à n'importe quel moment du développement sans entraîner de développement supplémentaire. Il s'agit en quelque sorte d'un prototypage évolutif en continue. Par la suite, les vues ont été créées et travaillées de façon collaborative avec les parties prenantes, de sorte que tout le monde s'entende sur le produit final. Des vues ont également été créées ou modifiées une fois que l'application fut mise en production. Cependant, afin d'entamer la vie en production de l'application avec toutes les possibilités nécessaires, il est intéressant d'inclure certains acteurs tôt dans le développement. Dans le cas de PPAL, ces deux étapes se sont effectuées de façon conjointe avec toute une panoplie des futurs utilisateurs de l'application et non uniquement les experts du domaine.

La plupart des fonctionnalités présentement implémentées dans OM sont le résultat des besoins des clients de PPAL. Un des avantages d'une méthodologie telle qu'OM est qu'au fur et à mesure du développement de nouvelles applications, les nouvelles fonctionnalités qui seront développées selon des besoins particuliers deviennent disponibles pour toutes les applications antérieures et postérieures utilisant la plateforme OM.

PPAL a été développé sur plusieurs années et au cours de celles-ci, différents stades de développement de l'application ont été présentés afin de s'assurer que l'interface, les fonctionnalités et les technologies utilisées soient acceptées autant par les clients, les futurs utilisateurs (techniciens, ingénieurs et chercheurs), les équipes de TI, les architectes TI, les gestionnaires de projets et les gestionnaires de budgets.

PPAL a été mis en production dans les systèmes d'entreprise par le département informatique d'HQ en décembre 2019. Cette mise en production a demandé un effort considérable. En plus de devoir passer plusieurs revues de code, plusieurs parties de l'application ont dues être retravaillées afin de se conformer aux lignes directrices de l'entreprise, aux exigences de sécurité et aux exigences de maintien des applications. Des tests d'acceptation auprès des clients d'affaires et des clients fonctionnels ont aussi dû être passés pour assurer le respect des diverses exigences.

De plus, des fonctions de sécurité pour empêcher l'injection malveillante de SPARQL ont dû être mise en place. Le reformatage de tous les journaux de l'application a dû être effectué pour que ceux-ci soient conformes au schéma de l'agrégateur de journaux de l'entreprise. Des fonctions pour assurer le typage des données ont dues être développées, les BDT étant plus permissives que les BDR et trop permissives pour les directives d'entreprise. Le cadre Spring permettant les services REST de l'application a dû être complètement revu pour s'adapter au nouveau cadre Open Shift (OpenShift, 2020) de l'entreprise. L'application a ainsi pu être intégrée dans une image Docker (Docker, 2020) gérée par Kubernetes (kubernetes, 2020). Des pipelines Jenkins (Jenkins, 2020) ont été utilisés pour effectuer le déploiement et les mises à jour de l'image.

Des tâches gérées par le serveur, comme le déclenchement quotidien de l'inférence et le nettoyage quotidien des individus vides, ont dues être reprogrammées en PL/SQL (en anglais *Procedural Language / Structured Query Language*) (PL/SQL, 2020) dans des tâches Control-M (Control-M, 2020). Les mécanismes de modifications des ontologies et modèles de données qui étaient codés en Java ont aussi dû être recodés en PL-SQL.

Afin de mettre l'application en production, une quantité importante de documentation a été réalisée. Parmi les documents notables se trouvent les suivants :

- Procédure pour apporter des modifications à l'ontologie (15 pages);
- Manuel d'utilisateur (28 pages);
- Guide des tests pour maintenance (190 pages) ;
- Formulaire de définition des niveaux de support;
- Formulaire de définition des cas d'utilisation;
- Formulaire de définition des besoins;
- Présentations de formations (plus de 120 diapositives).

Six formations ont eu lieu pour que l'auteur de la présente thèse forme quatre développeurs et une ingénieure de bases de données au fonctionnement de l'application et aux technologies utilisées. Trois formations ont aussi eu lieu pour former les différents utilisateurs à l'utilisation de l'outil.

De plus, les commentaires de l'entièreté du code ont été améliorés afin de se conformer au standard des Javadocs. Finalement, des commentaires à travers le code ont été mis en place afin d'expliquer le fonctionnement des technologies du WS, peu connues des développeurs qui devront maintenir l'application.

5.2 Solution développée

En substance, OM est un constructeur de vues mettant à profit les avantages des technologies du WS pour permettre de parcourir une ou des ontologies afin d'y découvrir les connaissances qui y sont contenues en croisant itérativement les données des différentes classes entre elles. C'est un outil Web qui permet à la fois l'exploration (trouver de nouvelles connaissances) et l'exploitation ontologique (utiliser les connaissances existantes). À la manière de DATE, OM se sert de requêtes génériques pouvant interroger le modèle de données en même temps que les données.

OM permet la création de vues afin d'échanger de l'information entre diverses parties prenantes. De plus, il permet aussi la modification des données des bases de connaissances

sous-jacentes. OM a été conçu pour être utilisé par des néophytes des technologies du WS et des langages de manipulation de données en général.

Les vues qui en résultent peuvent donc être partagées entre les utilisateurs et être utilisées pour ajouter, modifier ou effacer des données dans les bases de connaissances. L'approche utilisée entraîne l'auto-adaptabilité de ces vues aux changements du modèle de données. Ainsi, les modifications apportées aux modèles de données sont immédiatement visibles pour tous les utilisateurs, sans recompilation de l'application. Cette auto-adaptabilité est particulièrement utile dans un contexte de prototypage évolutif, d'approche agile ou dans un contexte de réingénierie constante tel que l'Industrie 4.0.

5.2.1 Vue d'ensemble

Comme pour les applications précédentes, OM a été développé selon une architecture trois-tiers. Pour arriver à cet outil, une nouvelle composante visuelle qui, à notre connaissance, n'est disponible dans aucune librairie, a dû être créée. Cette composante prend la forme d'une grille extensible représentant un graphe de données liées qui permet de naviguer à travers le modèle par la sélection de facettes sur les propriétés. Elle constitue le point central de l'application. L'espace de solution est ainsi construit itérativement et des fonctionnalités pour modifier la grille permettent la création intuitive de la vue souhaitée par l'usager. La Figure 5.1 montre le UI d'OM.

Comme dans le cas du CRV-HQ, la navigation débute par la sélection d'une classe dans l'arbre des classes du modèle. Les individus de cette classe ainsi que ses propriétés sont alors retrouvés par OM. Les propriétés de type données de la classe peuvent ensuite être utilisées pour appliquer des filtres qui prennent la forme de composantes dont les fonctionnalités varient en fonction du type de donnée. Les propriétés de type objets, elles, peuvent être utilisées pour à la fois explorer le modèle et façonner l'espace de solution. Une pléthore d'autres fonctionnalités viennent appuyer l'usager dans la construction de sa vue.

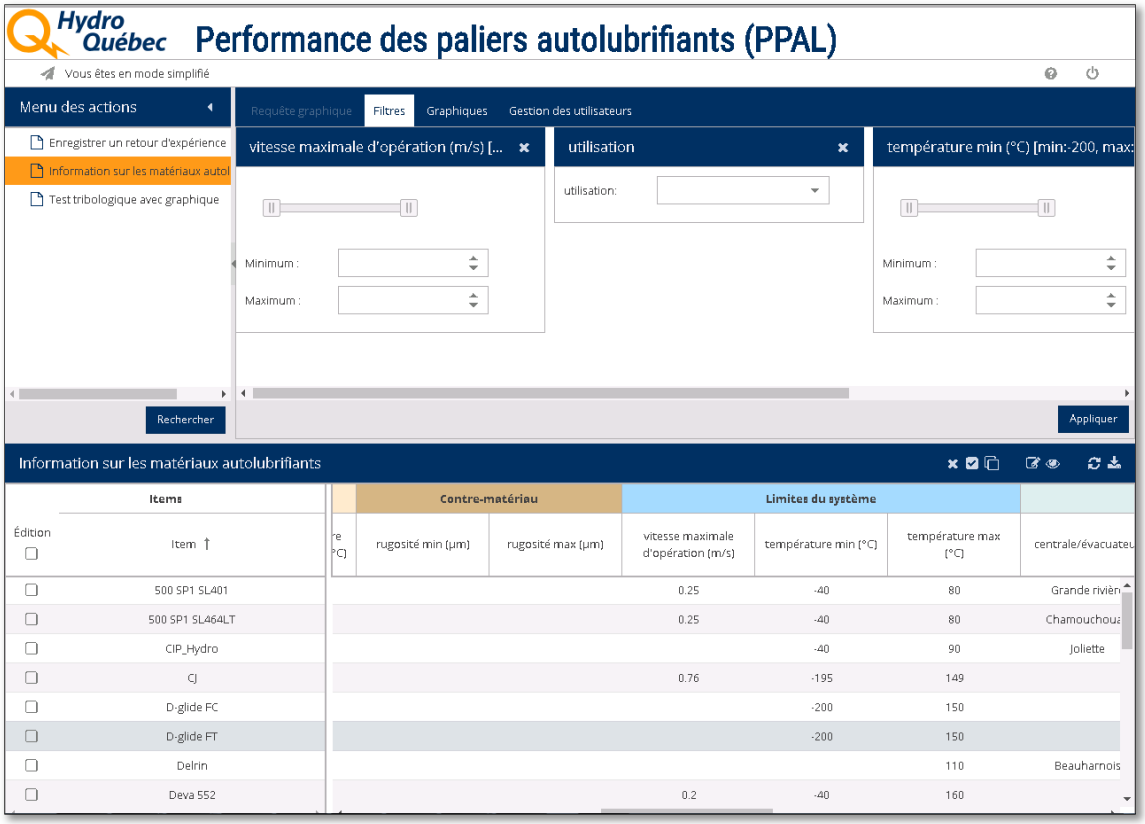

Figure 5.1 Vue d'ensemble d'OM

5.2.2 Séparation conceptuelle des graphes

Comme c'était le cas pour DATE et le CRV-HQ, le modèle et les données ont été divisés en plusieurs sous-graphes dans la BDT. Pour n'importe quelle application conçue avec OM, cinq graphes sont nécessaires : le TBox, le ABox, le EBox, le SBox et l'ontologie de visualisation (OV). Ces graphes sont jumelés par une agrégation virtuelle pour permettre à OM de fonctionner.

L'OV contient un schéma dont les propriétés (principalement d'annotations) servent à diriger l'adaptation de l'interface d'OM. Elle est agnostique de domaine. Ses sémantiques permettent de personnaliser certains composants Web pour des usages particuliers et de charger et sauvegarder les vues des utilisateurs. Elle est présentée plus en détails à la section Ontologie de visualisation et est disponible dans son intégralité dans l'ANNEXE III.

Le TBox contient le schéma des données d'un domaine en particulier, c'est-à-dire l'ontologie du domaine de l'application. Ses classes et ses propriétés peuvent être annotées avec des éléments de l'OV afin de spécifier des fonctionnalités d'interface.

Le ABox contient des données qui se conforment au TBox.

Le EBox est un type de graphe inventé pour les besoins d'OM qui contient les énumérations du TBox. Séparer ainsi le EBox des TBox et ABox permet de traiter les énumérations comme des données que peuvent modifier les utilisateurs d'OM de la même manière que n'importe quelle donnée du ABox. Cependant, celles-ci font conceptuellement partie du TBox et doivent être séparées du ABox pour permettre la création de nouvelles applications utilisant le même TBox mais un ABox différent. Les énumérations des ontologies utilisées par OM sont différentes de celles préconisées par le standard OWL. L'implémentation a été faite de cette manière pour pouvoir utiliser les mêmes fonctionnalités d'OM pour gérer les énumérations que pour gérer les données ordinaires. Cela dit, les versions subséquentes d'OM devraient tendre vers l'utilisation complète du standard OWL, afin de rendre la méthodologie plus universelle.

Le SBox contient les vues sauvegardées. Les sauvegardes peuvent contenir à la fois des sémantiques provenant du TBox, du ABox et du EBox et il est donc judicieux de les conserver dans leur propre sous-graphe.

5.2.3 Exploration ontologique

L'interface d'OM contient quatre divisions principales telles qu'on peut le voir dans la Figure 5.2. En haut se trouve une barre de tâches, à gauche un arbre des vues et des classes et dans la partie centrale supérieure se trouve un panneau à onglet. Celui-ci contient trois onglets : l'onglet de la requête graphique, l'onglet des filtres et l'onglet des graphiques. Finalement, la partie centrale inférieure présente la grille sémantique.

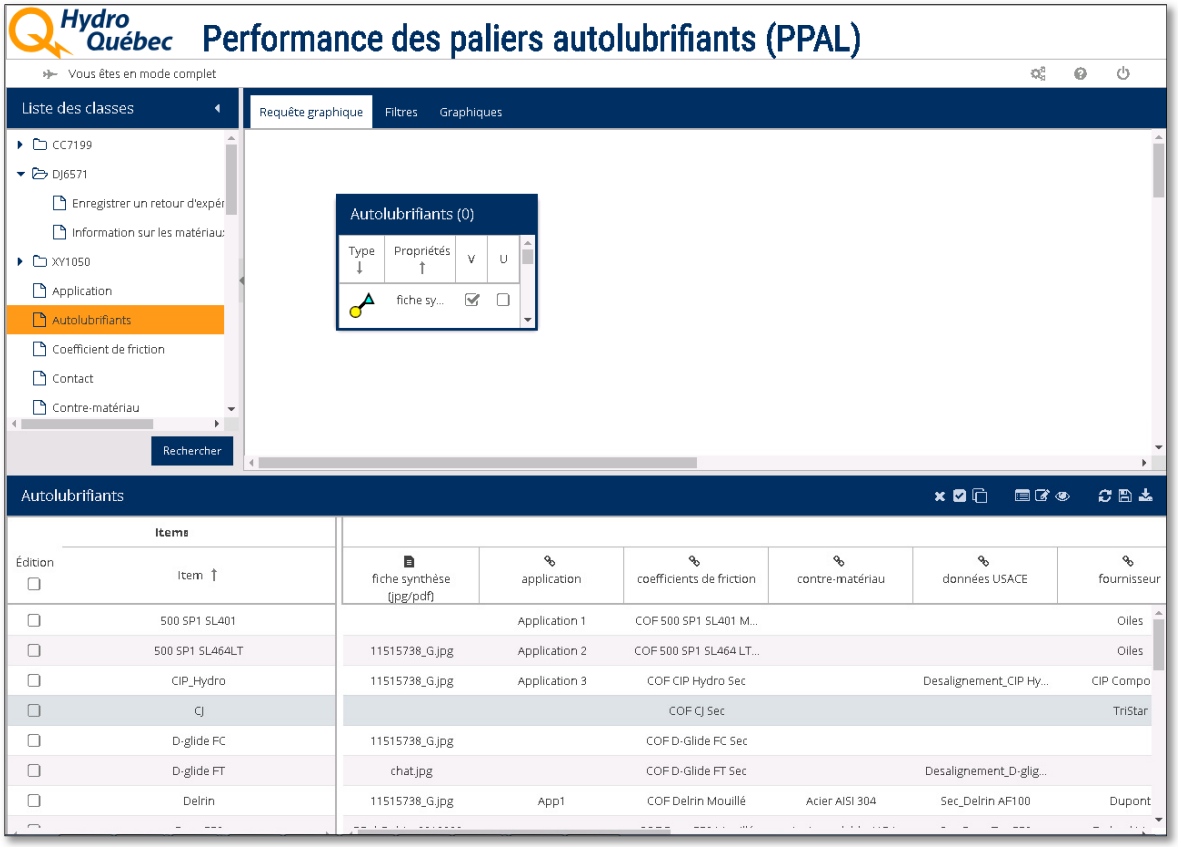

Figure 5.2 Divisions de l'interface

Tout comme pour le CRV-HQ, le point d'entrée d'OM est une liste des étiquettes (*rdfs:label*) des classes OWL (*owl:Class*) contenues dans l'ontologie de référence (voir Figure 5.3). Tel que pour le CRV-HQ, cet arbre ne permet que la sélection de la classe pivot, la première classe de la requête. Une différence notable entre ces arbres provient du fait que dans le cas d'OM les vues créées et sauvegardées grâce à l'outil sont présentées à l'intérieur de l'arbre, dans des répertoires aux noms des utilisateurs qui les ont créées.

Tout comme pour le CRV-HQ, l'exploration ontologique se fait en deux temps. La sélection d'une classe de la liste par l'utilisateur entraîne une requête sur la BDT, l'apparition d'une boîte avec les propriétés de cette classe dans l'espace de la requête graphique et le chargement des premiers individus de cette classe dans la grille sémantique. La deuxième phase de l'exploration ontologique peut ensuite se faire de deux manières, par l'outil de requête graphique ou par la grille sémantique.

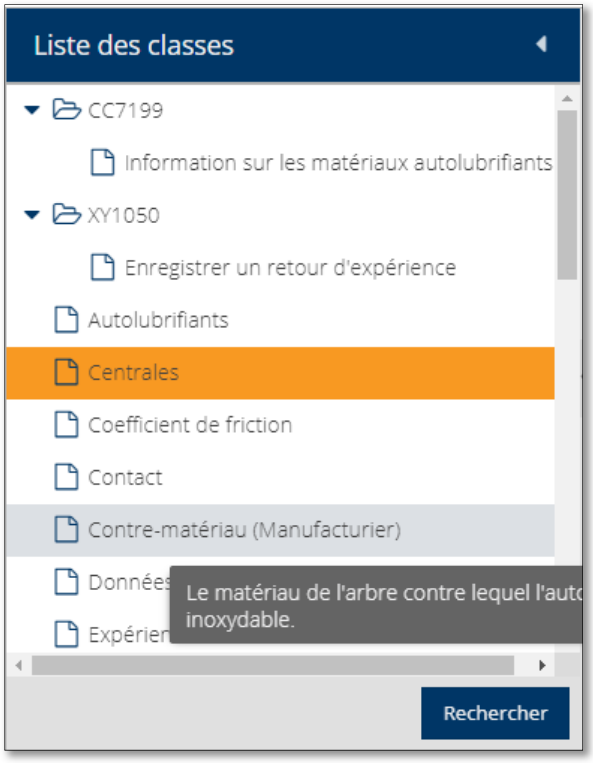

Figure 5.3 Liste des classes et des vues

5.2.3.1 Grille sémantique

La grille sémantique est le point central d'OM. Le graphe de la classe, de ses propriétés et de ses individus y est présenté sous forme d'un tableau en deux dimensions. Tel que mentionné, la première classe chargée depuis la liste des classes devient la classe pivot de la vue.

Cette classe pivot est le point de départ de la requête et toutes les autres classes qui feront partie de la vue lui seront nécessairement reliées. À partir de cette première classe, l'usager pourra naviguer dans le graphe pour construire sa vue à l'aide des propriétés de type objets. Les enjeux de l'utilisation de la classe pivot seront discutés en détails dans le chapitre suivant.

Les individus de la classe pivot sont conceptuellement les sujets de la requête et c'est pourquoi les noms de ceux-ci sont figés à gauche de la grille. Ainsi, si le nombre de colonnes de la grille devient assez grand pour nécessiter l'utilisation d'une barre de défilement horizontale, ces individus ne sont jamais perdus de vue.

La Figure 5.4 montre la séparation entre les individus de la classe pivot et le reste des données. Les étiquettes (*rdfs:label*) des individus de cette classe sont présentées sous la colonne « Item ». La colonne d'édition est aussi immobilisée à la gauche de la grille et ne sera pas non plus affectée par la barre de défilement horizontale. Partout dans l'application, le terme « item » a été préféré au terme individu, afin de faciliter sa compréhension par les utilisateurs néophytes des technologies sémantiques.

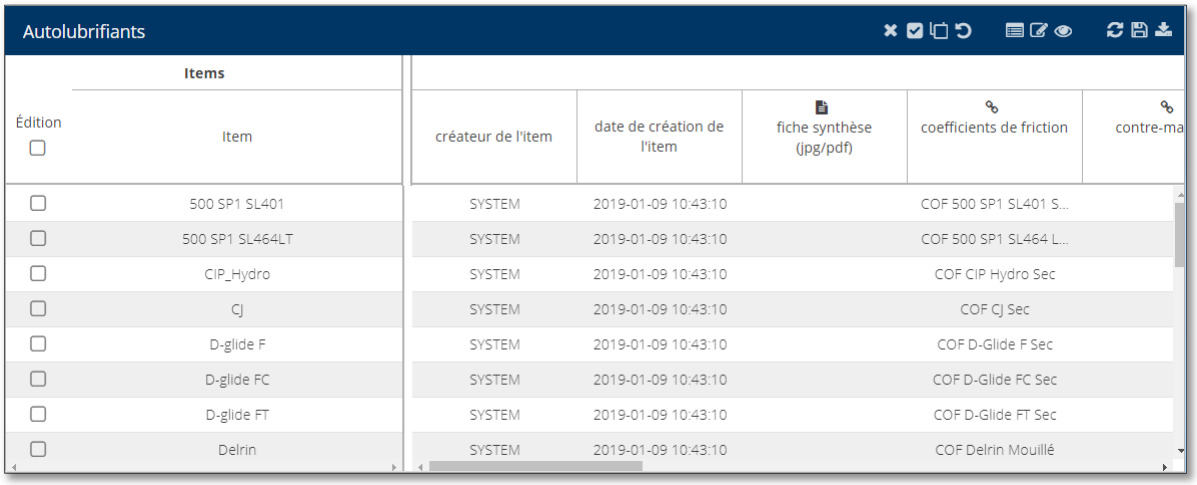

Figure 5.4 Grille sémantique

Les propriétés d'une classe sont utilisées pour constituer les colonnes de la grille sémantique. Ces colonnes sont donc les facettes de l'espace de solution. Toutes les propriétés présentées sont de deux types généraux mutuellement exclusifs : les propriétés de type données (*owl:DatatypeProperty*) et les propriétés de type objets (*owl:ObjectProperty*).

Les propriétés de type données sont des feuilles du graphe RDF qui contiennent des valeurs littérales. OM se sert des types de données (*datatypes*), modélisés dans le codomaine des propriétés de type données par *rdfs:range* afin de s'assurer de la présentation en bonne et due forme des valeurs littérales ainsi que pour adapter les composants de l'interface tels que les filtres ou les champs des formulaires.

Les propriétés de type données n'ont généralement aucune icône accompagnant leur étiquette puisqu'elles sont, pour ainsi dire, des colonnes normales d'une grille, n'ayant pas de fonctionnalité spéciale leur étant associée. Cela dit, certains contenus particuliers entraînent des fonctionnalités sur les cellules, tels que les URL, les chaînes de caractères trop longues ou les fichiers.

Les propriétés de type objets, quant à elles, sont des branches du graphe liant les individus d'une classe aux individus d'une autre classe. Dans la Figure 5.4, les classes ayant l'icône en forme de lien ( ) sont des propriétés de type objets. Une fonctionnalité permet à l'utilisateur de charger les propriétés décrivant la classe codomaine de cette propriété (définie par *rdfs:range*) dans la grille ainsi que les valeurs des individus qui sont liées aux individus de la classe pivot. La Figure 5.5 et la Figure 5.6 illustrent cette fonctionnalité. Dans la Figure 5.5, on voit la classe pivot avant l'utilisation de cette fonctionnalité et dans la Figure 5.6, on voit la classe pivot et la classe ajoutée.

C'est de cette façon que l'usager peut parcourir les différentes branches du graphe et ainsi explorer l'ontologie. En parcourant les liens entre les classes, l'usager compose implicitement une requête dont il voit constamment l'espace de solution.

On peut voir dans la partie supérieure de l'espace de travail de la Figure 5.5 et de la Figure 5.6, la requête graphique qui se compose de la même manière que dans le CRV-HQ. En effet, l'exploration ontologique se fait de manière semblable en parcourant les liens d'association entre les classes, ce qui a permis la réutilisation de ces composantes Web. Cependant, le mécanisme de chargement sous-jacent est complètement différent.

Dans le cas du CRV-HQ les objets graphiques sont transformés en une requête SPARQL unique qui est ensuite lancée contre la base de données. Ici, ce sont des requêtes génériques qui interrogent la base de données classe par classe. Les résultats sont ramenés en utilisant des véhicules génériques (un par classe) d'une manière semblable au mécanisme présenté dans DATE. La grille est ensuite reconstruite dans le client à partir des données de ces objets génériques.

C'est la nature sémantique des données qui permet le chargement des classes de la vue indépendamment les unes des autres. En effet, la grille est populée en utilisant les URI afin

d'aligner les individus des classes non-pivots aux individus de la classe pivot. Ce chargement paresseux au niveau de la classe rend possible d'utiliser le même ensemble de requêtes génériques pour permettre l'ajout *ad hoc* d'une classe à l'ensemble de solution.

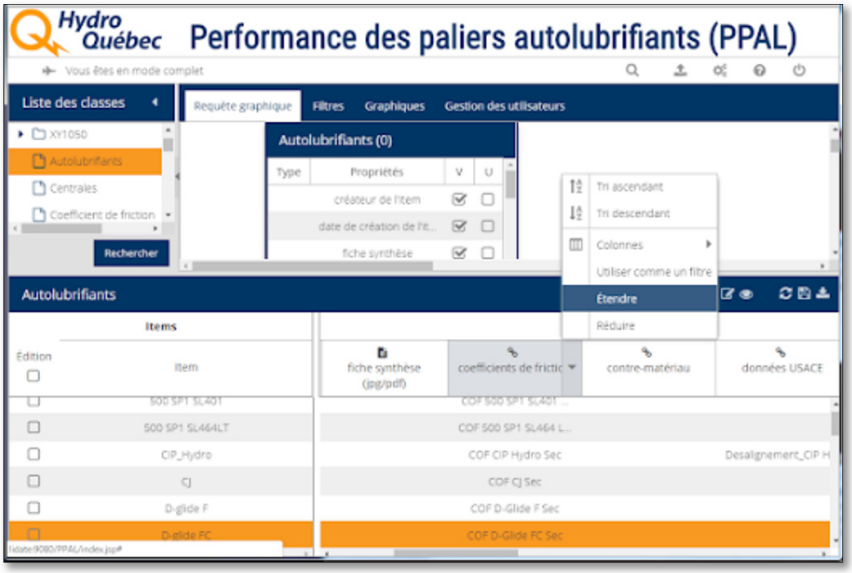

Figure 5.5 Classe pivot

Figure 5.6 Classe étendue

Puisque nous sommes dans un graphe, il peut arriver qu'un individu de la classe pivot pointe vers de multiples individus de la classe suivante. Lorsque c'est le cas, le nombre de liens possibles est inscrit dans la case comme on peut le voir dans la Figure 5.7.

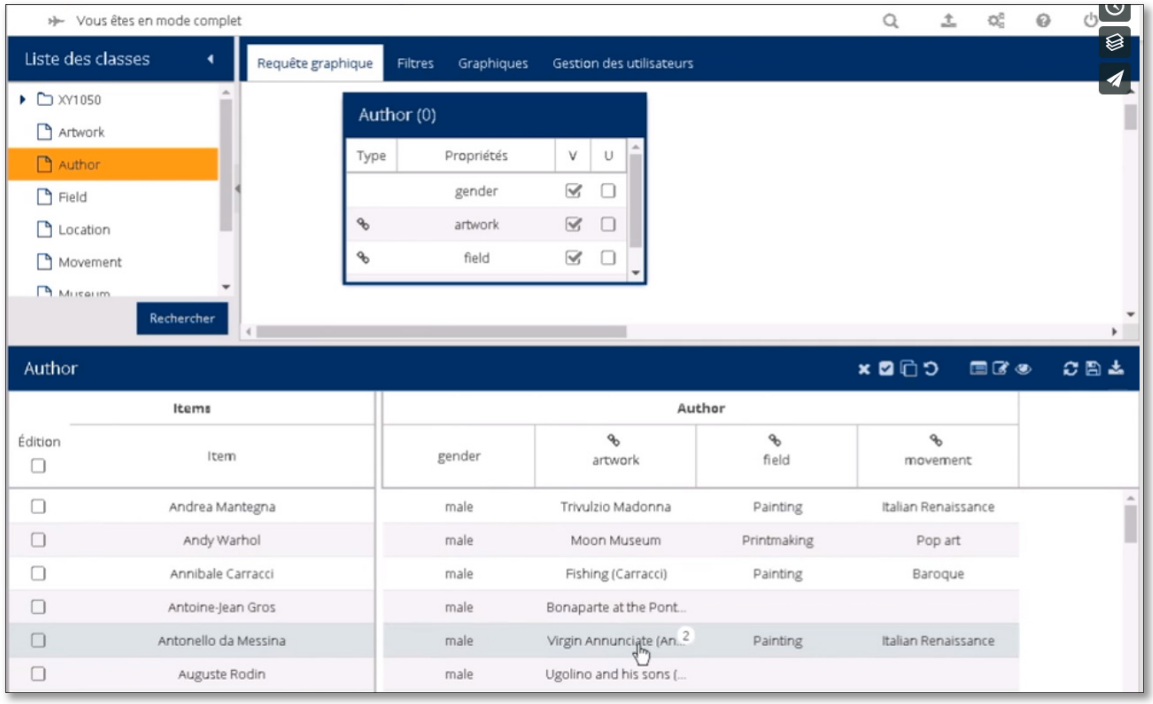

Figure 5.7 Nombre de liens possibles

En cliquant dans une cellule d'une colonne de type objets pointant vers plusieurs individus, un menu contextuel s'affiche contenant les choix de liens possibles, c'est-à-dire les étiquettes (*rdfs:label*) des individus de la classe de codomaine qui sont liés à l'individu de la classe actuelle par cette propriété. La sélection d'un de ces liens fait changer la valeur de cette case ainsi que les valeurs des cellules de cette ligne contenues dans les colonnes des classes suivantes, selon le graphe de cet individu particulier. Un exemple est visible à la Figure 5.8. Dans cette figure la ligne CIP_Hydro a été copiée. Les valeurs des cellules sous la colonne « contact » sont différentes dans les deux cas, ainsi on peut voir que les valeurs des colonnes « fonction » et « nom du contact » ont été modifiées en conséquence.

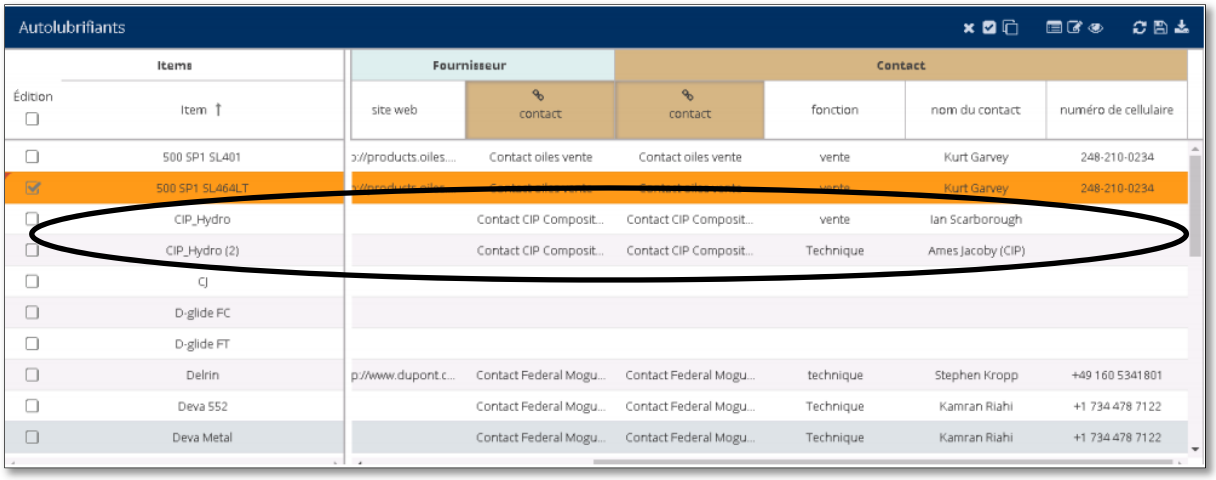

Figure 5.8 Changement d'une valeur d'une cellule de type objet

Ce mécanisme réconcilie le format multidimensionnel du graphe avec le format en deux dimensions de la grille. Grâce à lui et à l'outil de copie de lignes (voir la section Les outils sur les lignes de l'ANNEXE II) on s'assure que l'usager puisse composer tous les espaces de solutions rendus possibles par la forme en graphe des données, à l'intérieur d'une grille.

L'entête de la grille sémantique comporte deux niveaux. Le niveau supérieur présente l'étiquette (*rdfs:label*) des classes tandis le niveau inférieur présente les étiquettes des propriétés de cette classe. Lorsque l'on ajoute une classe liée à une autre dans la grille, l'entête de la propriété objet (sur le niveau inférieur) faisant le pont entre les deux classes devient d'une couleur particulière et l'entête de la classe liée (sur le niveau supérieur) devient de la même couleur. De cette façon, l'usager peut facilement comprendre quelle colonne contient les noms des individus de cette classe. Il peut donc aussi savoir rapidement quelles valeurs de la grille changeront s'il change la valeur d'une colonne objet.

Voici un exemple pour récapituler. On peut apercevoir dans la Figure 5.9 la colonne « limites de systèmes ». Cette colonne a l'icône « 🔗 » qui montre qu'il s'agit d'une colonne de type objet. Cette colonne, appartenant conceptuellement à la classe des Matériaux, présente des individus de la classe des « Limites du système (Manufacturier) » caractérisant ces matériaux. Chaque matériau pouvant être associé à plusieurs jeux de limites de systèmes, chaque cellule de la colonne « limites de systèmes » peut être cliquée pour afficher les différents jeux de

limites associés à un matériau. Les matériaux étant les sujets de cette requête, chacun occupe une ligne de la grille. Si un matériau a deux jeux de limites de systèmes, deux options seront présentées lorsque l'usager appuiera sur la cellule de la ligne de ce matériau sous la colonne « limites de systèmes ». Le choix d'un jeu par rapport à l'autre engendrera la modification des valeurs des cellules des propriétés de la classe des « Limites du système (Manufacturier) ». Autrement dit, pour l'exemple de la Figure 5.9, la modification de la valeur de la cellule au croisement de la ligne orange et de la colonne dont l'entête inférieure est turquoise engendre la modification des valeurs des autres cellules de la ligne orange dont l'entête supérieure de la grille a la même couleur turquoise.

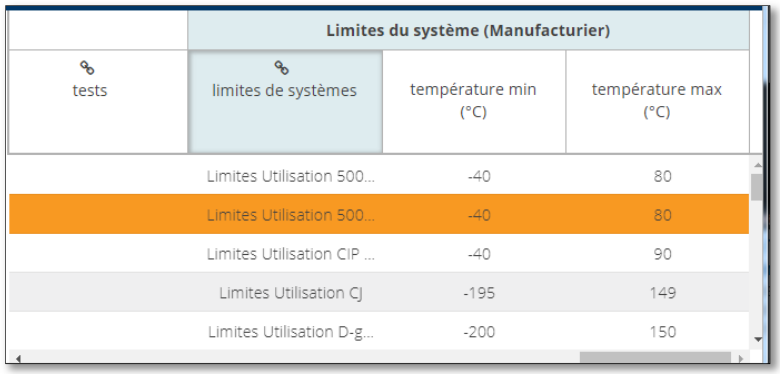

Figure 5.9 Entête de la grille

Chaque colonne contient un menu contextuel d'actions pouvant être utilisées sur celle-ci (voir Figure 5.10). Ce menu contextuel permet d'étendre et de réduire une colonne, de trier la grille, de cacher ou de montrer une colonne ou encore d'appliquer un filtre sur la colonne afin de réduire l'espace de solution. Cette dernière fonctionnalité fait apparaître un composant de filtre dans l'onglet des filtres du panneau central. Ce composant s'adapte selon le codomaine ou le type de la propriété (*rdfs:range* ou *rdf:type*) et sera donc différent s'il s'agit d'une date, d'un nombre, d'une chaîne de caractère ou d'un objet.

Figure 5.10 Menu des outils sur les colonnes

En haut à droite de la grille sémantique, on retrouve 10 boutons qui servent à actionner différentes fonctionnalités de la grille (voir Figure 5.11). Ces boutons sont séparés en trois groupes distincts. Les quatre boutons à gauche sont des outils servant à manipuler les lignes de la grille. Les trois boutons centraux permettent des fonctions d'édition et de visualisation en formulaire d'une ligne de la grille. Les trois boutons de droites permettent de rafraîchir, de sauvegarder et d'exporter la grille.

Figure 5.11 Outils de la grille sémantique

Les outils sur les lignes permettent de copier, conserver, cacher ou réafficher les lignes. Ils sont présentés en détails dans l'ANNEXE II, tout comme les outils pour rafraîchir, sauvegarder ou exporter la grille.

Les outils d'édition et de visualisation en formulaire permettent d'apporter des modifications à la BDT. Tous trois font apparaître une fenêtre contenant une vue en formulaire des données, construite dynamiquement. Ce formulaire contient les données d'une ligne sélectionnée dans la grille et permet donc de visualiser puis d'ajouter, d'enlever ou de modifier les valeurs des propriétés des individus des différentes classes présents sur cette ligne de la grille.

Ces trois outils sont très semblables (voir Figure 5.12, Figure 5.13 et Figure 5.15) Ils ouvrent une fenêtre contenant autant de formulaires qu'il y a de classes dans la grille. La principale différence entre les deux premiers outils est que le premier présente un onglet par classe et que le second présente tous les formulaires dans un seul et même onglet. La principale différence entre le troisième outil et les deux premiers, est que celui-ci permet uniquement de visualiser les données alors que les deux autres permettent l'utilisation des formulaires pour modifier la base de connaissances.

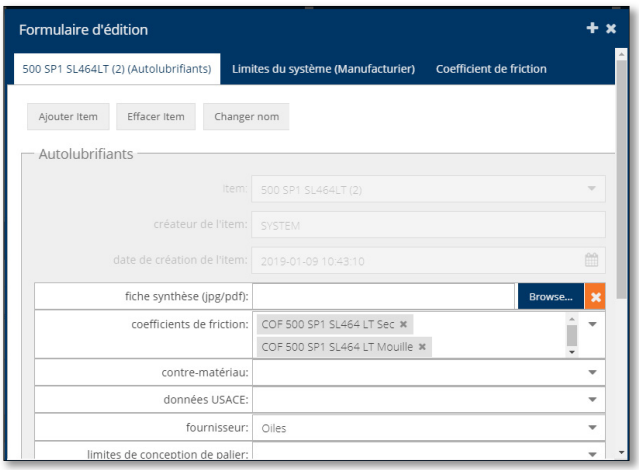

Figure 5.12 Formulaire CRUD à onglet

De façon similaire aux filtres le type des champs de chaque propriété dépend du codomaine de la propriété (*rdfs:range*) et ceux-ci seront donc de type numérique, chaîne de caractère, date, date-temps, fichier ou objet et auront les fonctionnalités associées à ce type de champ. De plus, à l'instar des filtres, les annotations effectuées sur le TBox à l'aide des propriétés de l'OV permettent de restreindre les valeurs acceptées.

Les seules cardinalités pour l'instant implémentées dans OM sont celles impliquées par la propriété *owl:FunctionalProperty*. Cette sémantique OWL est utilisée pour signifier qu'une propriété relie un individu d'une classe à un maximum d'une valeur ou d'un individu. Par exemple, la propriété 'père' pourra être fonctionnelle reliant un individu (le fils) à au maximum

un autre individu (son père). De même, la propriété ‘âge’ devrait être fonctionnelle puisque les personnes n’ont en général qu’un seul âge.

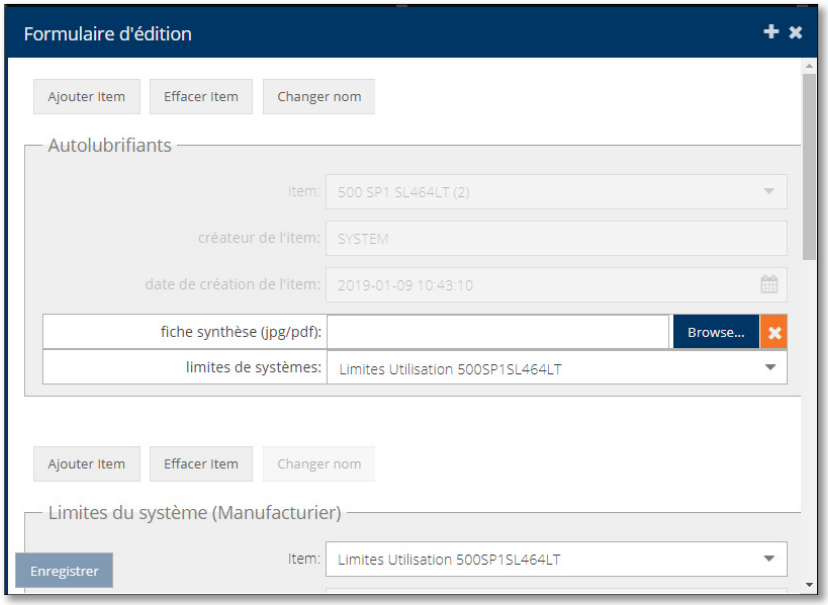

Figure 5.13 Formulaire CRUD simplifié

Les propriétés de type objets qui sont définies comme fonctionnelles dans le TBox, seront donc représentées par des champs *combobox* ne permettant qu’une seule valeur possible à la fois. Les propriétés de type objets qui ne sont pas fonctionnelles, quant à elles, seront représentées par des champs *tagfield* permettant d’entrer autant de valeurs que le désire l’utilisateur. On peut voir ces deux types de champs à la Figure 5.14.

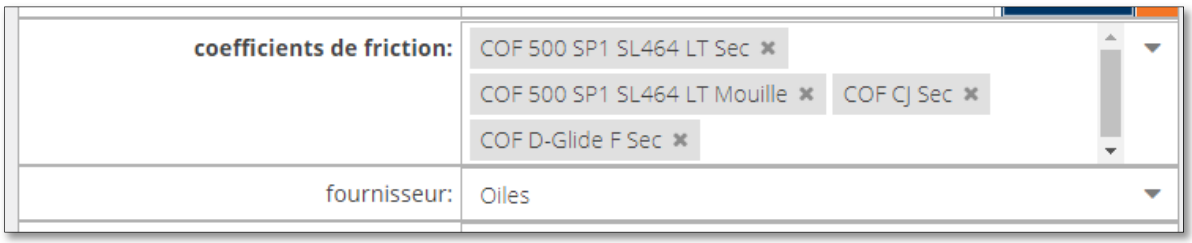

Figure 5.14 Combobox et Tagfield

Vue la nature de graphe des données, un champ d'un formulaire peut être lié à un champ d'un autre formulaire, par des relations d'association ou encore par des relations d'association inverses. Des mécanismes ont dû être mis en place pour que la modification d'un champ par l'usager entraîne automatiquement la modification de tous les champs concernés ainsi que pour vérifier que la cohérence logique de la BDT soit préservée.

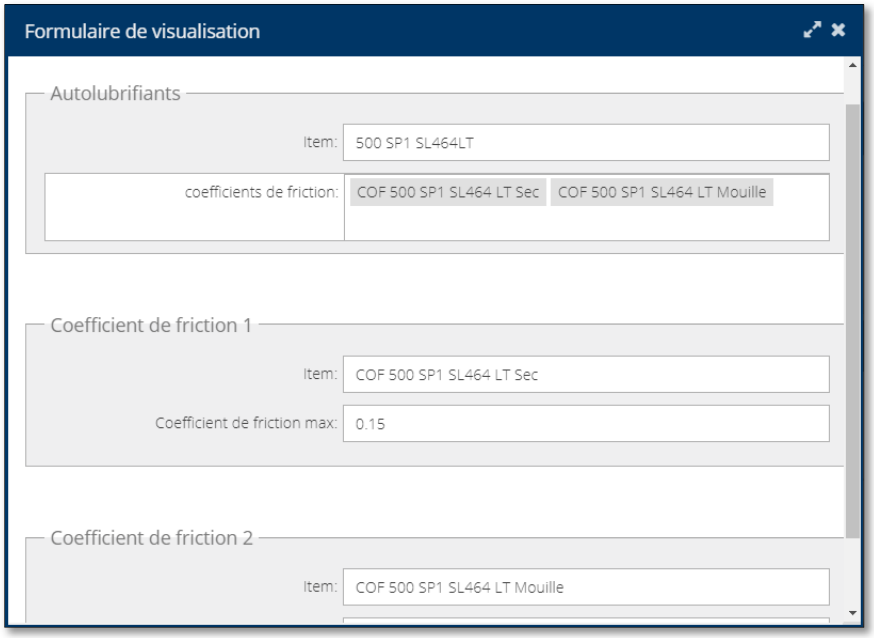

Figure 5.15 Visualisation du formulaire

5.2.3.2 Onglet de la requête graphique

Cet onglet contient une version graphique de la requête qu'est en train de construire l'usager (voir Figure 5.16). Conçu à partir des mêmes composantes que le CRV-HQ, il permet à l'usager, s'il le préfère, d'utiliser ces composantes graphiques pour explorer l'ontologie et construire l'espace de solution. Comme pour le CRV-HQ, chaque boîte représente une classe. À l'intérieur de celles-ci sont représentées les propriétés de type données (sans icône) et les propriétés de type objets de la classe (avec icônes de lien). Les propriétés de subsomption n'ont pas été implémentées. L'utilisateur pourra utiliser ces boîtes afin d'afficher, de cacher,

d'étendre ou de réduire des colonnes de la grille sémantique ou encore d'ajouter ou d'enlever des filtres à la vue.

L'utilisateur a donc deux mécanismes d'interactions distincts pour construire la vue. La grille sémantique et l'onglet de requête graphique sont liés programmatiquement et l'utilisation de l'un entraînera des modifications réciproques sur l'autre.

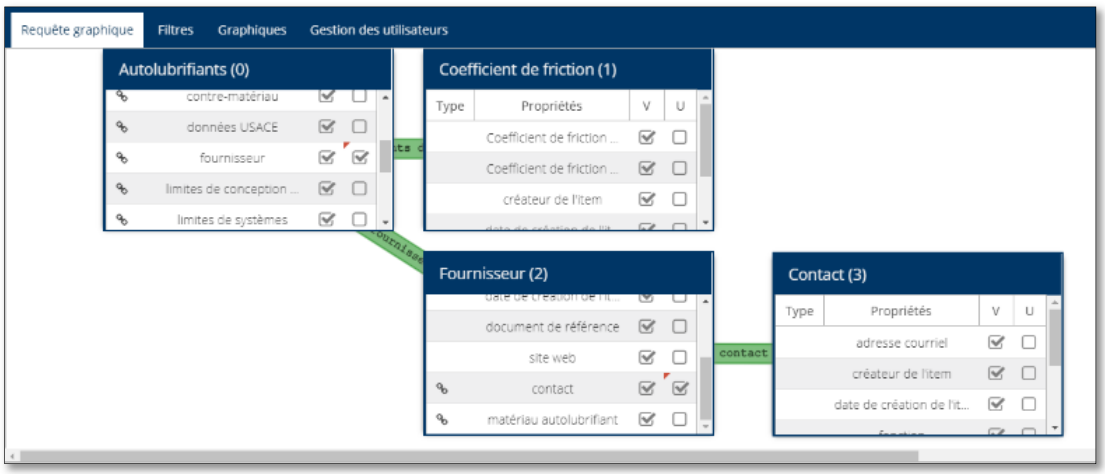

Figure 5.16 Requête graphique d'OM

5.2.4 Réduction de l'espace de solution

L'usager peut, à partir de la grille ou de l'outil de requête graphique, mettre des filtres sur les propriétés de l'ontologie de domaine afin de réduire l'espace de solution (voir Figure 5.17).

Ainsi cinq types de filtres correspondant aux types des codomaines des propriétés vont pouvoir être ajoutés à l'onglet des filtres : les filtres numériques, les filtres dates et date-temps, les filtres chaînes de caractères et les filtres objets.

Les filtres numériques, dates et dates-temps permettront à l'utilisateur d'ajouter des bornes supérieures ou inférieures aux valeurs des propriétés sélectionnées. L'OV pourra être utilisée pour imposer des valeurs minimales et maximales à ces bornes et ainsi orienter les choix de l'utilisateur. Les filtres de chaînes de caractères fonctionnement quant à elles avec des champs textes comparant les expressions régulières aux données de la propriété dans la BDT.

Finalement, les filtres objets fournissent dans un menu contextuel les valeurs présentes dans la BDT pour une propriété de type objet.

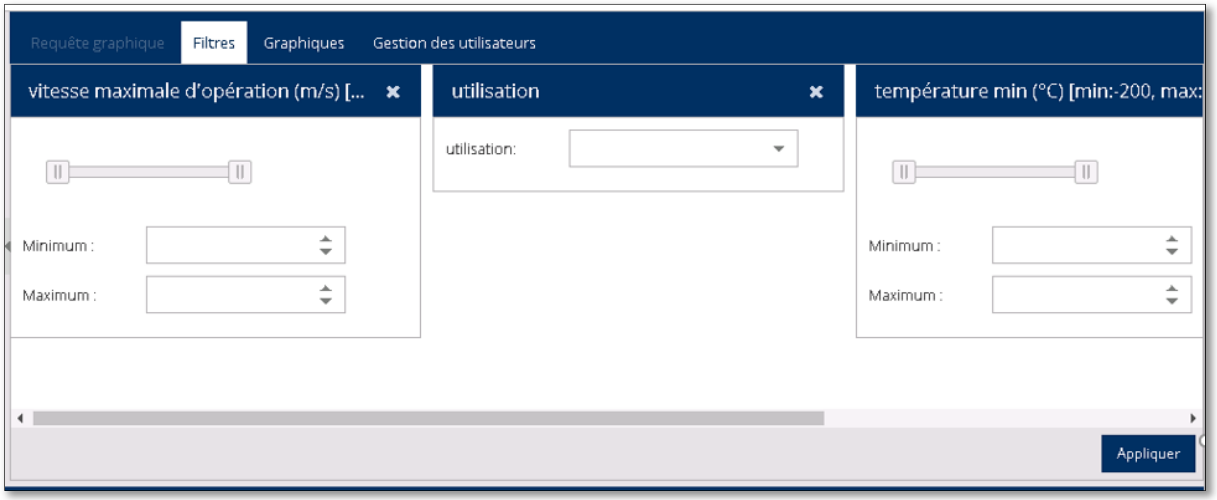

Figure 5.17 Types de filtres

Tous ces filtres peuvent être nuancés grâce à une option qui permet de retrouver les individus répondant à ces filtres ainsi que les individus où il n'existe pas de triplets pour la propriété en question. Cette option est utile, puisque dans un monde ouvert, on ne peut pas déduire de l'absence d'information que l'individu ne répond pas au filtre. Par exemple, c'est le cas si on veut évaluer les options de pièces pouvant résister à une certaine température et que certains des fabricants fournissent la température maximale alors que d'autres ne la fournisse pas. Il peut alors être intéressant d'obtenir les pièces dont on ne connaît pas la valeur recommandée de température maximale dans l'espace de solution afin de poser la question au fournisseur.

Cette option de filtrage des valeurs nulles vient en deux saveurs : le filtrage superficiel et le filtrage profond. En premier lieu, on peut vouloir voir tous les individus de la classe pivot qui sont liés à un individu de la classe de la propriété filtrée répondant aux critères du filtre ou dont la valeur de la propriété filtrée est inconnue. En deuxième lieu, on peut vouloir voir tous les individus de la classe pivot qui sont liés ou non à un individu de la classe de la propriété filtrée répondant aux critères du filtre ou dont la valeur de cette propriété est inconnue, sachant que si l'individu n'est pas lié à cette classe, la valeur de la propriété filtrée est nécessairement

inconnue. Dans ce deuxième cas, il en va de même pour toute autre classe liant la classe pivot et la classe de la propriété filtrée.

Par exemple, considérons le patron de classes X -> Y -> Z. On cherche tous les individus de X qui sont liés à un individu de Y qui lui est lié à un individu de Z qui lui possède une propriété pz dont la valeur se situe entre 0 et 1.

Sans filtrage des valeurs nulles, la requête ramènera uniquement les individus de X liés à un individu de Y, lui-même lié à un individu de Z dont la propriété pz se situe entre 0 et 1.

Dans la première saveur qu'est le filtrage superficiel des valeurs nulles, la requête ramènera tous les individus de X liés à un individu de Y, lui-même lié à un individu de Z dont la propriété pz se situe entre 0 et 1 et tous les individus de X liés à un individu de Y, lui-même lié à un individu de Z dont la propriété pz n'est pas évaluée.

Dans la deuxième saveur, le filtrage profond des valeurs nulles, la requête ramènera les mêmes valeurs que dans la première saveur, en plus des individus de X liés à un individu de Y qui, lui, n'est pas lié à un individu de Z et des individus de X qui ne sont pas liés à un individu de Y.

Un exemple concret pourra apporter un éclairage plus intuitif à ces options de filtrages des valeurs nulles. Posons que l'on cherche les matériaux (X) dont le fournisseur (Y) a publié un catalogue (Z) qui peut montrer la température maximale (pz) à laquelle il peut être soumis.

Une requête sans filtrage des valeurs nulles ramènera tous les matériaux dont le fournisseur a publié un catalogue dans lequel la température maximale se situe entre 0 et 100 °C.

Une requête avec filtrage superficiel des valeurs nulles ramènera tous les matériaux dont le fournisseur a publié un catalogue dans lequel la température maximale se situe entre 0 et 100 °C ou dans lequel la température maximale n'est pas indiquée.

Une requête avec filtrage profond des valeurs nulles ramènera tous les matériaux dont le fournisseur a publié un catalogue dans lequel la température maximale se situe entre 0 et 100 °C ou dans lequel la température maximale n'est pas indiquée, tous les matériaux dont le fournisseur n'a pas publié de catalogue et tous les matériaux dont on ne connaît pas le fournisseur.

Puisque dans un monde ouvert, on ne peut déduire de l'absence d'une information que cette information n'existe pas, cette deuxième saveur permet d'être certain d'obtenir tous les résultats qui répondent à la requête en plus de tous les résultats qui pourraient potentiellement répondre à la requête.

5.2.5 Mode d'utilisation

L'application contient deux modes distincts : le mode simple et le mode complet. Puisque le mode complet sert à créer les vues qui seront consommées en mode simple, c'est ce mode qui a été présenté depuis le début de ce chapitre. Il permet à l'utilisateur d'utiliser l'ensemble des fonctionnalités développées pour OM. Cependant, seuls les usagers ayant les droits nécessaires, typiquement des super-utilisateurs, peuvent accéder au mode complet.

Le mode simple a été créé afin de protéger les utilisateurs normaux d'OM de la complexité émanant de ses fonctionnalités les plus avancées destinées aux super-utilisateurs. Il est en fait un sous-ensemble des fonctionnalités du mode complet. Il sert en outre à faciliter la gestion des droits de créations et de modifications des vues.

Le mode simple est donc très semblable au mode complet mais avec moins de fonctionnalités. L'utilisateur y est confiné à parcourir les vues créées par les super-utilisateurs et partagées en mode simple. Ces vues se retrouvent dans le menu des actions qui remplacent la liste des classes. En mode simple, l'utilisateur ne peut pas étendre ou réduire les classes, utiliser l'onglet de requête graphique, sauvegarder, modifier ou supprimer une vue, lancer le moteur d'inférence ou encore télécharger des données par lot. Il peut en revanche appliquer ou changer des filtres sur les données, cacher ou montrer des colonnes, cacher ou copier des lignes, créer des graphiques sur les données de la grille et modifier des informations dans la base de connaissances à l'aide des formulaires CRUD (s'il possède les permissions nécessaires).

Le mode simple vise à orienter les utilisateurs dans l'exploration des données de la BDT et à leur permettre des modifications ciblées sur celle-ci. Dans l'application PPAL, le mode simple contient deux actions. La première action sert à orienter les ingénieurs dans la formulation de leurs requêtes pour comparer des matériaux autolubrifiants entrant dans la conception des

équipements des centrales électriques. La deuxième action est destinée aux ouvriers qui font l'installation, la maintenance et le retrait des pièces fabriquées avec les matériaux autolubrifiants. Dans le premier cas, les utilisateurs sont des experts du domaine et ne sont pas vraiment concernés par des paramètres comme la rapidité d'utilisation. En revanche, dans le deuxième cas, il est crucial que l'outil soit simple, rapide d'utilisation et garde l'usager à l'abri des erreurs. Les utilisateurs cibles de la première action auront probablement les droits d'accès au mode complexe, où la vue servira de point de départ pour leur recherche. Les utilisateurs cibles de la deuxième action seront quant à eux confinés au mode simple qui est mieux adapté à leur besoin.

5.2.6 Construction de la grille sémantique

Une boucle permet la construction de la grille sémantique et de son contenu à partir des graphes de chaque classe qui sont indépendants les uns des autres. Ces graphes reçus en format JSON-LD sont tout d'abord cadrés (*framed*) et compactés grâce à la librairie JSON-LD.js. Chaque graphe est alors stocké dans une page d'un magasin du cadre de développement Web Ext.js. Pour construire la grille, le tableau (*array*) des colonnes sera d'abord construit selon l'ordre des classes et des propriétés qu'a choisi l'usager. En l'absence d'ordre choisi par l'usager : (1) les classes seront mises dans l'ordre qu'elles se présentent dans la requête, (2) leurs propriétés de type données seront ensuite ajoutées en ordre alphabétique, puis (3) leurs propriétés de type objets seront finalement ajoutées en ordre alphabétique.

Une fois les colonnes construites, une boucle à trois niveaux permet de peupler la grille. Cette boucle itère, classe par classe, sur chaque propriété et sur chaque individu. Les individus de la classe pivot sont tous présentés par défaut en ordre alphabétique tandis que les individus des autres classes dépendent des valeurs présentes dans les cellules de la propriété de type objet de la classe qui les précède et qui fait le pont entre les deux classes. Sa complexité est discutée dans la section Complexité d'OM du CHAPITRE 6.

5.2.7 Manifestation multiple des classes

OM se sert extensivement des URI dans toutes ses fonctionnalités. OM permet aux usagers de naviguer l'ontologie en suivant les liens des propriétés de type objets. L'usager peut donc créer une vue qui contient plusieurs fois la même classe. Prenons l'exemple suivant : on désire connaître les personnes qui connaissent des personnes qui connaissent une personne. Le patron de cette requête sera alors le suivant :

ex:Personne --ex:connait--> ex:Personne --ex:connait--> ex:Personne

où ex:Personne est une classe et --ex:connait--> est une propriété de type objet. Plusieurs fonctionnalités de la grille nécessitent de connaître exactement dans laquelle de ces trois classes effectuer une action particulière. Dans un tel cas, l'URI de la classe à lui seul n'est pas suffisant pour savoir si l'action doit être effectuée sur la première, la deuxième ou la troisième manifestation de la classe ex:Personne. On utilise alors le chemin (*path*) qui va de la classe pivot à la classe observée afin de départager les différentes manifestations d'une même classe.

Deux types de chemins sont utilisés dans l'application. Le chemin des classes et le chemin des propriétés. Toute l'application aurait pu être construite en utilisant uniquement le chemin des propriétés, mais cela aurait augmenté sensiblement la complexité du code.

Le chemin des classes correspond à la liste des classes précédant cette classe, dans leur ordre d'apparition depuis la classe pivot. Dans l'exemple précédent, la classe pivot est ex:Personne et son chemin est alors une liste vide : []. La deuxième manifestation de la classe ex:Personne a comme chemin : [« ex:Personne »], tandis que la troisième manifestation a comme chemin: [« ex:Personne », « ex:Personne »].

Le chemin des propriétés correspond à la liste des prédicats liant cette classe à la classe pivot, dans leur ordre d'apparition. Dans l'exemple précédent, la classe pivot est ex:Personne et son chemin est alors une liste vide : []. La deuxième manifestation de la classe ex:Personne a comme chemin: [« ex:connaît »], tandis que la troisième manifestation a comme chemin: [« ex:connaît », « ex:connaît »].

Les chemins sont donc des listes d'URI de propriétés ou de classes. Le chemin des propriétés permet d'identifier de façon unique n'importe quelle classe d'une requête, peu importe le

nombre de fois où elle est manifestée dans une requête. Ce principe prend pour hypothèse que les propriétés objets n'ont qu'un seul codomaine dans l'ontologie avant inférence. Cette hypothèse est cruciale au bon fonctionnement de l'application et sera discutée en détail dans la section Inférence.

On notera que le chemin des classes n'identifie pas à lui seul de manière unique chaque manifestation possible d'une classe puisque deux propriétés d'une même classe pourraient avoir le même codomaine. Par exemple la classe des Personnes pourrait avoir la propriété Mère et la propriété Père dont les codomaines sont tous deux la classe des Personnes.

Le mécanisme des chemins permet ainsi d'identifier toutes les colonnes de la grille de façon unique grâce à deux informations : l'URI de la propriété et le chemin des propriétés ayant mené à sa classe.

Toutes les cellules de la grille peuvent donc aussi être identifiées de façon unique à partir de trois informations : l'URI de l'individu de la classe pivot (qui permet d'identifier de manière unique la ligne), l'URI de la propriété de cette colonne, et le chemin des propriétés ayant mené à la classe de cette propriété. Afin d'identifier de manière unique les cellules des lignes copiées, il est nécessaire d'affubler aux lignes copiées un nouvel URI unique. Celui-ci est généré automatiquement.

5.2.8 Sauvegardes

L'utilisateur peut sauvegarder une vue par le biais d'une fenêtre contextuelle de sauvegarde (voir Figure 5.18). Il pourra y nommer la vue, lui attribuer un commentaire, la partager en mode simple et appliquer différentes permissions à lui associer en fonction des rôles possibles des utilisateurs.

Pour l'instant, OM offre trois types de permission. La permission de consultation signifie que les usagers ayant un certain rôle peuvent consulter la vue, mais ne peuvent pas en modifier les individus. La permission de modification indique que les usagers peuvent consulter et modifier tous les individus de la vue. La permission de modification restreinte indique que les usagers peuvent consulter la vue, mais ne peuvent que modifier les individus dont ils sont le créateur.

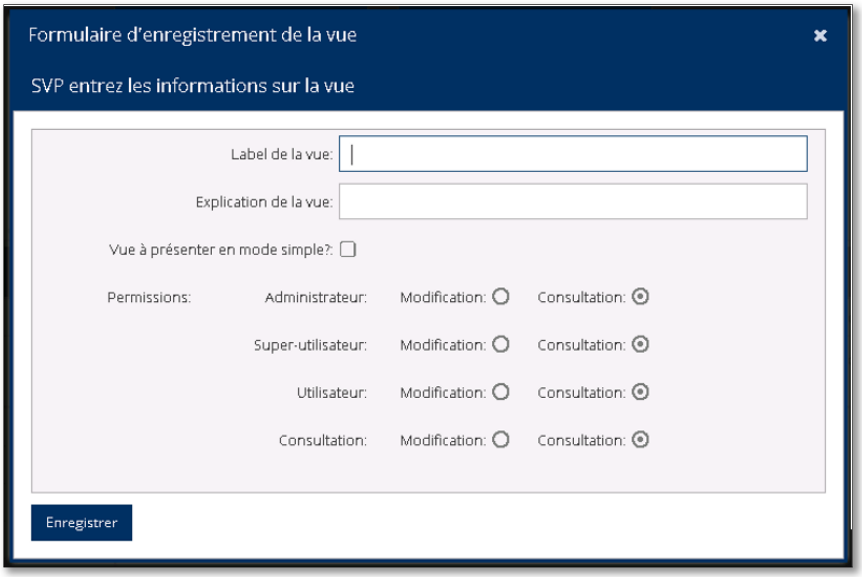

Figure 5.18 Formulaire de sauvegarde

L'OV contient deux propriétés de types données qui ont comme domaine *owl:Thing*, c'est-à-dire qui, par inférence, sont des propriétés de toutes les classes de l'ontologie de domaine. La première indique l'identifiant de l'usager qui a créé chaque individu du ABox et la deuxième indique l'heure et la date de cette création. Ces deux informations sont enregistrées systématiquement et automatiquement à la création de chaque individu. Ce sont grâce à ces propriétés que la permission modification restreinte peut être appliquée.

Lorsque l'usager enregistre une vue, un URI est d'abord généré puis toutes les informations permettant de recréer la grille dans le même état avec les mêmes valeurs sont alors trouvées et utilisées dans la création d'un graphe RDF en JSON-LD. Le graphe formé par ces informations est alors envoyé au serveur pour qu'il soit sauvegardé dans la BDT.

5.2.9 Inférences

OM a été conçu avec l'objectif de tirer avantage des moteurs d'inférences et de la puissance des langages RDFS et OWL. Les moteurs d'inférences permettent en effet de déduire de l'information implicitement contenu dans les données grâce à l'expressivité de RDFS et

d'OWL. Chacun de ces langages vient avec un ensemble d'implications (*entailment*) permettant de valider la cohérence logique des assertions dans le graphe et de déduire des assertions supplémentaires.

Lors du développement, des tests ont été faits avec des moteurs d'inférences permettant le chaînage avant et le chaînage arrière. Le chaînage arrière implique généralement que l'inférence est appliquée à chaque requête, uniquement sur les ressources concernées par cette requête. Le chaînage avant, lui, est appliqué sur la BDT au complet, déduisant toutes les inférences possibles avant que les requêtes ne soient effectuées. À chaque itération de l'engin de chaînage avant, les règles d'inférence ne produisant plus de nouvelles informations sont retirées, jusqu'à ce que toutes les règles soient retirées ou que des conditions supplémentaires d'arrêt soient atteintes.

Dans une application comme OM, où les usagers peuvent constamment modifier les données du modèle, et éventuellement le modèle lui-même, le chaînage arrière semble s'avérer le choix pertinent. Le problème avec le chaînage avant dans le contexte d'OM, est qu'il oblige à refaire l'inférence sur l'ensemble du jeu de données après chaque modification ou suppression de triplets. L'ajout de triplets, quant à lui, n'entraîne pas l'obligation de refaire l'inférence sur l'entièreté du jeu de données, puisque des engins comme celui d'Oracle permettent dans ce cas une fonctionnalité d'inférence incrémentale qui infère uniquement les connaissances qui peuvent être déduites des nouveaux triplets et les ajoute à l'ensemble d'inférences existant. Cela dit, le temps d'attente lors des opérations de suppression ou de modification des données devient alors prohibitif, même sur des jeux de données relativement petits.

Le chaînage arrière en contrepartie aurait permis d'effectuer les inférences à la pièce et il en aurait résulté un temps d'attente un peu plus long lors des lectures mais considérablement moindre lors de l'écriture. Cependant, l'utilisation d'Oracle ayant été une contrainte d'entreprise durant cette recherche et le moteur d'inférence d'Oracle ne permettant pas le chaînage arrière, OM a été développé pour fonctionner en chaînage avant.

Deux artifices ont donc dû être développés pour ne pas refaire l'inférence à chaque suppression ou modification des données. Il a ainsi été possible de réaliser tous les cas d'utilisations rencontrés jusqu'à ce jour en mettant à profit le chaînage avant.

Le premier artifice consiste à faire l'inférence sur l'ensemble du jeu de données une seule fois par jour, en période creuse. Cette façon de faire implique que des vérifications doivent être effectuées à chaque transaction pour éviter que des incohérences logiques ne soient découvertes que trop tard. Certaines informations implicites ne sont alors pas ajoutées immédiatement aux résultats, ce qui entraîne le besoin pour le second artifice.

Le second artifice consiste à programmer certaines règles d'inférence jugées cruciales pour le bon fonctionnement d'OM, et d'utiliser ces fonctions en temps réel, directement dans l'application. Par exemple, dans OM les relations inverses (*owl:inverseOf*) sont persistées dans le modèle et leur modification est assurée par la programmation, sans utiliser l'engin d'inférence. De cette façon, l'engin d'inférence n'a pas à être sollicité à chaque fois qu'un usager enlève ou modifie un triplet. La validation des relations inverses est importante pour préserver la cohérence logique de la base de connaissances, surtout lorsqu'elles sont utilisées conjointement avec les cardinalités fonctionnelles (*owl:functional*). Puisqu'OM utilise de manière extensive ces deux sémantiques, il est apparu nécessaire de valider et d'ajuster les relations inverses à chaque transaction.

Une autre limitation du moteur d'inférence d'Oracle est qu'il ne supporte qu'un sous-ensemble des règles d'OWL 2. Entre autres, il ne permet pas de construire des classes avec restrictions où l'on peut trouver les individus dont les valeurs sur une certaine propriété se trouvent dans une certaine fourchette de valeur. Les tests fait avec Stardog le permettaient ce qui permettait aussi d'enregistrer les vues des utilisateurs en tant que classes avec restrictions directement dans le modèle. Cette limitation a donc eu des implications sur l'implémentation de la solution. La première implication est que les sauvegardes se font sous formes de graphes RDF et n'utilisent en aucun cas l'inférence pour déterminer quels individus font partis de l'espace de solution. La deuxième implication est que les filtres doivent être appliqués autrement que par l'inférence. Dans les deux cas, l'application s'éloigne d'un style d'implémentation qui aurait pu être plus près des technologies du WS.

5.2.10 Paradigmes de classification et d'héritage

L'interface d'OM présente les classes d'une ontologie et des propriétés associées à ces classes de façon que les utilisateurs puissent comprendre la nature des individus de cette classe et ajouter des individus s'ils le désirent. Le paradigme de classification dans lequel on retrouve les jeux d'inférences RDFS et OWL et le paradigme d'héritage des langages OO s'opposent dans la façon dont les propriétés sont associées aux classes dans le modèle.

Dans un paradigme d'héritage, les propriétés applicables à une classe sont les propriétés de cette classe et de ses superclasses.

Dans un paradigme de classification, on déduit le type d'un individu en fonction de ses propriétés et des classes qui ont comme domaine ces propriétés. On pourra ainsi déduire si un individu est un organisme vivant, un mammifère ou un humain en trouvant les propriétés qui sont associées à la sous-classe la plus spécifique de la hiérarchie. Dans ce contexte, lorsqu'on tente de trouver les propriétés qui ont comme domaine une certaine classe, après que les jeux d'inférences RDFS et OWL aient été utilisés sur les données, on retrouvera les propriétés ce cette classe et de ses sous-classes.

Ce constat peut sembler surprenant et c'est pourquoi il sera maintenant présenté plus en détails.

On retrouve ce mécanisme à l'intérieur des règles rdfs2, rdfs9 et rdfs11 de l'ensemble d'inférence RDFS (RDFS, 2020) (voir Tableau 5.1) et des règles scm-dom1 et prp-dom des sémantiques du vocabulaire de schéma OWL 2 (OWL, 2020) (voir Tableau 5.2).

Tableau 5.1 Règles 2, 9 et 11 de RDFS

	If	**then**
rdfs2	?p1 rdfs:domain ?c1 . ?i1 ?p1 ?i2 .	?i1 rdf:type ?c1 .
rdfs9	?c1 rdfs:subClassOf ?c2 . ?i1 rdf:type ?c1 .	?1 rdf:type ?c2 .
rdfs11	?c1 rdfs:subClassOf ?c2 . ?c2 rdfs:subClassOf ?c3 .	?c1 rdfs:subClassOf ?c3 .

Tableau 5.2 Règles scm-dom1 et prp-dom d'OWL

	If	**then**
scm-dom1	?p1 rdfs:domain ?c1 . ?c1 rdfs:subClassOf ?c2 .	?p1 rdfs:domain ?c2 .
prp-dom	?p1 rdfs:domain ?c1 . ?i1 ?p1 ?i2 .	?i1 rdf:type ?c1 .

On voit dans le Tableau 5.1 par la règle rdfs2, que si une propriété p1 a comme domaine une classe c1 et que cette propriété relie l'individu i1 à l'individu ou la valeur i2, on peut alors déduire que l'individu i1 est de type c1.

Par la règle rdfs9, on voit que si la classe c1 est une sous-classe de la classe c2 et que l'individu i1 est de type c1, on peut déduire que i1 est aussi de type c2.

Par rdfs11, on voit que si la classe c1 est une sous-classe de la classe c2 et que c2 est une sous-classe de c3, alors c1 est une sous-classe de c3.

À partir de ces trois règles, posons qu'une propriété p1 a comme domaine une classe c1, qu'une propriété p2 a comme domaine une classe c2, qu'une propriété p3 a comme domaine une classe c3 et que la classe c1 est une sous-classe de c2 qui est une sous-classe de c3.

Posons un individu i2 de type c2. On peut déduire des règles présentées plus haut qu'il est aussi de type c3, mais pas de type c1.

Dans le Tableau 5.2, on voit que par scm-dom1, si une propriété p1 a comme domaine la classe c1 et que la classe c1 est une sous-classe de c2, alors la propriété p1 a aussi comme domaine la classe c2.

Dans notre exemple, la propriété p1 a comme domaine, après inférence, les classes c1, c2 et c3. Ceci implique que si l'on regarde le type le plus spécialisé de l'individu i2, on trouve qu'il est c2. Si ensuite on essaie de déterminer quelles propriétés ont comme domaine la classe c2, on trouve la propriété p1 et la propriété p2. Cependant, la propriété p3 n'a comme domaine que la classe c3. Nous sommes donc bel et bien dans un paradigme fondamentalement différent de l'héritage OO.

C'est que nous sommes ici dans un paradigme de classification. Prenons maintenant un exemple concret. Un homme (c1) est un humain (c2) qui est un mammifère (c3). Si nous

sommes en présence d'un humain, celui-ci peut avoir des caractéristiques d'un homme ou d'une femme. Tant que ces caractéristiques spécialisées sont inconnues nous ne sommes en présence que d'un être humain.

En effectuant une recherche sur un individu particulier dont le type est inconnu, on peut trouver toutes les propriétés qui le caractérisent, tel que des propriétés spécifiques au genre, e.g., un humain qui est enceinte sera une femme. C'est pourquoi les propriétés des sous-classes sont applicables aux superclasses. En revanche, si l'on trouve un individu qui a des poils, cette propriété ne nous aidera pas à classifier l'individu comme étant un homme ou une femme, ni comme étant un être humain puisque beaucoup de mammifères peuvent avoir des poils. Afin de ne pas discriminer des types qui pourraient être celui de l'individu, la propriété ne doit pas descendre des superclasses vers les sous-classes.

Cela dit, une fois que l'on connait le type le plus spécialisé d'un individu, on pourra déduire ses super-types et leurs propriétés. Cela implique en revanche une recherche par les individus et non par le modèle.

Dans OM, on veut pouvoir à partir d'une classe déterminer toutes les propriétés qu'un individu de cette classe pourrait posséder, i.e. on veut que les propriétés de la classe et de ses superclasses soient affichées, et on veut pouvoir les déduire sans passer par les individus. Cependant une requête comme la Requête 5.1, ne permettra pas d'y parvenir directement sans réconcilier d'abord les paradigmes de classification et OO.

```
1       SELECT ?p
2       WHERE { ?p rdfs:domain ex:UneCertaineClasse}
```

Requête 5.1 Trouver les propriétés ayant comme domaine une certaine classe

La situation est semblable en ce qui concerne les codomaines des propriétés, par les règles rdfs3, rdfs9, rdfs11, scm-rng1 et prp-rng. Dans un paradigme de classification, une propriété se retrouve après inférence avec plusieurs codomaines : son codomaine original et toutes les superclasses de ce codomaine. Du côté client, la version actuelle d'OM nécessite qu'on

différencie le codomaine original et les codomaines inférés pour déterminer quelle classe doit être ramenée lorsque l'usager décide d'explorer les liens d'une propriété de type objet. Ce codomaine original est donc la classe la plus spécialisée de tous les codomaines, en prenant pour acquis qu'il n'y a qu'un seul codomaine original.

En résumé, OM nécessite de trouver les propriétés des superclasses de la classe sélectionnée et ce avec des requêtes uniquement sur le modèle puisqu'il permet d'explorer les données liées en se servant de l'ontologie et non des individus. Effectuer des requêtes sur les individus impliquerait qu'une propriété qui n'a jamais été utilisée ne trouverait pas son chemin jusqu'au UI. En outre, il semble préférable que l'interface présente une vision OO d'une classe (i.e. les propriétés de cette classe et de ses super-classes), puisque les propriétés des super-classes peuvent être nécessaires pour comprendre la nature d'un individu.

Deux solutions ont été envisagées pour obtenir des visions OO des propriétés des classes. La solution non retenue est présentée au CHAPITRE 6.

La solution qui a été utilisée dans OM consiste à faire inférer un jeu de règles avant d'utiliser les ensembles d'inférences RDFS et OWL. Ce jeu de règles de pré-inférences est exposé dans le Tableau 5.3. On y utilise le terme *directDomain* pour indiquer les domaines d'une propriété dans une perspective d'héritage OO et le terme *directRange* pour indiquer le codomaine le plus spécialisé des codomaines. De cette manière, on s'assure que (1) chaque propriété n'a qu'un seul *directRange*, que (2) ce *directRange* correspond à la plus spécialisée des classes de la hiérarchie pouvant être ce codomaine, que (3) chaque propriété à comme *directDomain* la classe observée et toutes ses superclasses et que (4) les propriétés inverses (qui n'ont pas encore été traitées puisque les jeux d'inférences RDFS et OWL ne sont utilisés qu'après) soient considérées dans l'attribution des *directDomain*. On peut par la suite procéder à l'inférence par les ensembles de règles RDFS et OWL.

Cette solution a été implémentée pour des ontologies ne définissant qu'un seul domaine et qu'un seul codomaine pour chaque propriété, avant inférence. Elle pourra cependant être adaptée pour pouvoir être appliquée aux ontologies ne respectant pas cette assomption en fonction de mécanismes développés pour adapter l'interface à cette problématique. Le

principal avantage de procéder par inférence pour définir les domaines et les codomaines directs est l'accélération obtenue lors de l'exécution des requêtes génériques.

Tableau 5.3 Ensemble de pré-inférences

	If	then
pi-rng	T(?p, rdfs:range, ?c)	T(?p, om:hasDirectRange, ?c)
pi-rng-inv-1	T(?p1, owl:inverseOf ?p2) T(?p2, rdfs:domain, ?c)	T(?p1, om:hasDirectRange, ?c)
pi-rng-inv-2	T(?p1, owl:inverseOf ?p2) T(?p1, rdfs:domain, ?c)	T(?p2, om:hasDirectRange, ?c)
pi-dom-1	T(?p, rdfs:domain ?c)	T(?p, om:hasDirectDomain, ?c)
pi-dom-2	T(?c1, rdfs:subClassOf+, ?c2) T(?p, rdfs:domain ?c2)	T(?p, om:hasDirectDomain, ?c1)
pi-dom-inv1	T(?c1, rdfs:subClassOf+, ?c2) T(?p1, rdfs:range, ?c2) T(?p1, owl:inverseOf, ?p2)	T(?p2, om:hasDirectDomain, ?c1)
pi-dom-inv2	T(?c1, rdfs:subClassOf+, ?c2) T(?p1, rdfs:range, ?c2) T(?p2, owl:inverseOf, ?p1)	T(?p2, om:hasDirectDomain, ?c1)

Après l'utilisation de l'ensemble de pré-inférences, une requête comme la Requête 5.2 ramènera alors les propriétés d'une classe et de ses superclasses. Les sémantiques *hasDirectDomain* et *hasDirectRange* peuvent donc être utilisées dans les requêtes génériques.

```
1    SELECT ?p
2    WHERE { ?p om:hasDirectDomain ex:UneCertaineClasse}
```

Requête 5.2 Trouver les propriétés par le domaine direct

5.2.11 Partie serveur

Le côté serveur d'OM est relativement léger compte tenu de la complexité de l'application. Cela est dû au fait que le serveur n'est qu'un lieu transitoire permettant de ramener l'information de la BDT vers le client et vice versa. Aucune classe Java ne cartographie la base

de données, le modèle et les informations de visualisation, puisque toutes les informations transitent sous forme de graphe entre la BDT et le client.

Outre les mécaniques de configuration, de journalisation, de création de session HTTP et de création de connexion et de modèles avec la BDT, OM ne contient qu'une collection relativement restreinte de services REST.

5.2.11.1 Services de construction de la grille

Lorsque l'usager demande de retrouver l'information sur une classe, l'interface Web appelle les services de construction de la grille. Le système détermine s'il doit ramener une ou plusieurs classes et la(es)quelle(s), puis s'il y a des filtres à appliquer. Les classes seront ramenées une par une dans des graphes génériques, à la manière de DATE. Pour chaque classe, le système retrouvera un graphe d'informations sur la classe, un autre sur ses propriétés et un autre sur ses individus contenant pour chacun d'eux les valeurs des propriétés.

De façon à accélérer le système, l'opération de construction du graphe des informations sur la classe, ses propriétés et ses individus a été séparée en deux étapes. On ramènera tout d'abord un sous-ensemble du graphe complet qui contiendra les triplets nécessaires, mais sans toutefois les classer. Celui-ci sera stocké en mémoire et c'est donc en mémoire que sera effectué le classement de ces triplets en un véhicule générique.

Ce sous-ensemble est retrouvé grâce à quatre requêtes. La première ramène tous les triplets ayant pour sujet la classe demandée (voir Requête 5.3).

```
1       CONSTRUCT {
2               <RANGE> ?c1 ?c2 .
3       }
4        WHERE {
5               <RANGE> ?c1 ?c2 .
6       }
```

Requête 5.3 Trouver les informations sur une classe

La deuxième requête (voir Requête 5.4) ramène toutes les propriétés dont le domaine ou le domaine-direct est la classe demandée. Ici, un choix a été fait et il sera discuté plus en détail au CHAPITRE 6.

```
1       CONSTRUCT {
2               ?prop ?pred ?obj
3       }
4       WHERE {
5               {
6                       ?prop uap:hasDirectDomain <RANGE>  .
7               }UNION{
8                       ?prop rdfs:domain <RANGE>
9               }
10              ?prop ?pred ?obj .
11      }
```

Requête 5.4 Trouver les informations sur les propriétés

La troisième requête (voir Requête 5.5) ramène les triplets concernant les individus du type de la classe demandée. Certains de ces triplets ont comme objet un URI. Ces URI sont les individus d'une autre classe. Comme l'usager ne demandera pas nécessairement de ramener les triplets concernant cette autre classe, mais qu'on nécessite les étiquettes de ces URI pour qu'elles soient présentées dans la grille, on réclamera ces étiquettes par la même occasion.

La Requête 5.5 présente la formulation générique de cette requête alors que la Requête 5.6 montre la formulation dans le cas où l'on recherche les individus d'une classe rattachés aux individus de la classe pivot par la propriété *ex:exemple*. Le système construira des requêtes légèrement différentes en utilisant le chemin des propriétés (voir section Manifestation multiple des classes) et les différents filtres appliqués sur cette branche de la requête. De plus, c'est dans cette requête qu'est appliqué l'outil de filtrage des valeurs nulles, superficiel ou profond, tel que décrit à la section Réduction de l'espace de solution. On viendra ajouter à cette dernière requête une limite (*LIMIT*) et un décalage (*OFFSET*) afin de faire un chargement paresseux sur les individus. Afin d'utiliser ces mots-clés, il est nécessaire d'utiliser une requête SELECT imbriquée dans la requête CONSTRUCT.

```
1   CONSTRUCT {
2       <varSelect> ?s ?iObjet .
3       ?iObjet rdf:type ?t .
4       ?iObjet rdfs:label ?lailp .
5       rdf:datatype ?dt ?iObjet .
6   }
7   WHERE
8   {
9       {
10          SELECT ?i0 WHERE {
11              <requeteIO>
12          } ORDER BY ?l0 LIMIT <LIMIT> OFFSET <OFFSET>
13      }
14      <requete>
15      <varSelect> ?s ?iObjet .
16      BIND (datatype(?iObjet) as ?dt)
17      OPTIONAL {
18          ?iObjet rdf:type ?t .
19          ?iObjet rdfs:label ?lailp .
20      }
21  }
```

Requête 5.5 Trouver les informations sur les individus : formulation générale

```
1   CONSTRUCT {
2       ?i1 ?s ?iObjet .
3       ?iObjet rdf:type ?t .
4       ?iObjet rdfs:label ?lailp .
5       rdf:datatype ?dt ?iObjet .
6   }
7   WHERE
8   {
9       {
10          SELECT ?i0 WHERE {
11              ?i0 ex:exemple ?i1 .
12          } ORDER BY ?l0 LIMIT <LIMIT> OFFSET <OFFSET>
13      }
14      ?i1 ?s ?iObjet
15      BIND (datatype(?iObjet) as ?dt)
16      OPTIONAL {
17          ?iObjet rdf:type ?t .
18          ?iObjet rdfs:label ?lailp .
19      }
20  }
```

Requête 5.6 Trouver les informations sur les individus : exemple de formulation

La quatrième et dernière requête (Requête 5.7) sert à ramener les propriétés d'annotation de l'OV. Cette façon de faire permet de ne pas avoir à modifier les requêtes du serveur lorsque

l'OV évolue. En ramenant systématiquement tous les triplets indiquant les relations de sous-propriétés aux trois super-propriétés de l'OV, on peut ainsi généraliser l'écriture de la requête qui fabrique le graphe à envoyer au client, comme nous le verrons dans les Requête 5.8 à Requête 5.11 .

```
1       CONSTRUCT {
2               ?annProp rdfs:subPropertyOf om:propDeClasse .
3               ?annProp rdfs:subPropertyOf om:propDeProp .
4               ?annProp rdfs:subPropertyOf om:propDeViz .
5       } WHERE {
6               {
7                       ?annProp rdfs:subPropertyOf om:propDeClasse .
8               } UNION {
9                       ?annProp rdfs:subPropertyOf om:propDeProp .
10              } UNION {
11                      ?annProp rdfs:subPropertyOf om:propDeViz .
12              }
13      }
```

Requête 5.7 Ramener l'ontologie de visualisation en mémoire

Les quatre graphes issus de ces requêtes sont ensuite chargés dans le même modèle de triplets en mémoire. Ce modèle contient tous les triplets nécessaires au bon fonctionnement de l'application. On veut maintenant donner une forme particulière au graphe pour faciliter son traitement par la partie client. On interrogera donc le modèle en mémoire avec la requête présentée dans les Requête 5.8 à Requête 5.11.

```
1  CONSTRUCT {
2      uap:reponse rdf:type uap:reponse .
3      uap:reponse uap:classe CLASSE  .
4      uap:reponse uap:props ?s .
5      uap:reponse uap:instances ?i .
6      CLASSE rdfs:label ?label .
7      CLASSE rdf:type owl:Class .
8      CLASSE uap:pathSize PREDPONDLENGTH
9      PATHQUERY
10     PATHRANGEQUERY
11     CLASSE rdfs:comment ?commentC . "
12     CLASSE ?pdvC ?vpdvC . "
13     ?s rdfs:label ?l .
14     ?s rdf:type ?t .
15     ?s rdfs:range ?r .
16     ?s owl:inverseOf ?io .
17     ?s rdfs:comment ?commentP .
18     ?s ?pdvP ?vpdvP .
19     ?s ?s ?val .
20     ?s ?s ?val2 .
21     ?i uap:instance ?i .
```

Requête 5.8 Construction du graphe générique (entête)

```
25  WHERE {
        //CLASS
26      OPTIONAL {
27              CLASSE rdfs:label ?label2 .
28              FILTER (langMatches(lang(?label2),LANGTAG))
29              BIND(STRDT(STR(?label2), xsd:string) AS ?label )
30      }
31      OPTIONAL {
32              CLASSE rdfs:comment ?commentC2 .
33              FILTER (langMatches(lang(?commentC2),LANGTAG))
34              BIND(STRDT(STR(?commentC2), xsd:string) AS ?commentC)
35      }
36      OPTIONAL {
37              CLASSE ?pdvC ?vpdvC .
38              ?pdvC rdfs:subPropertyOf uap:propDeClasse .
30              FILTER (?pdvC NOT IN (uap:propDeProp) ) .
40      }
```

Requête 5.9 Construction du graphe générique (corps - classe)

```
        //PROPERTIES
41      {
42              SELECT ?s ?r ?io ?t ?l ?func ?commentP ?pdvP ?vpdvP ?val ?val2
43              WHERE{
44                      {
45                              ?s uap:hasDirectDomain CLASSE  .
46                      }UNION{
47                              ?s rdfs:domain CLASSE
48                      }
49                      OPTIONAL{
50                              ?s rdfs:label ?l2 .
51                              FILTER (langMatches(lang(?l2),LANGTAG))
52                              BIND(STRDT(STR(?l2), xsd:string) AS ?l) "
53                      }
54                      OPTIONAL{
55                              ?s rdfs:comment ?commentP2 .
56                              FILTER (langMatches(lang(?commentP2),LANGTAG))
57                              BIND(STRDT(STR(?commentP2), xsd:string) AS ?commentP)
58                      }
59                      ?s uap:hasDirectRange ?r .
60                      ?s rdf:type ?t .
61                      OPTIONAL {
62                              ?s ?pdvP ?vpdvP .
63                              ?pdvP rdfs:subPropertyOf uap:propDeProp .
64                              FILTER (?pdvP NOT IN (uap:propDeProp) ) .
65                      }
66                      OPTIONAL{?s owl:inverseOf ?io . }
67                      OPTIONAL{?s rdf:type owl:ObjectProperty. BIND(owl:Anything as ?val2) . }
68                      OPTIONAL{?s rdf:type owl:DatatypeProperty . BIND('anything' as ?val) . }
69
70                              }
71      }
```

Requête 5.10 Construction du graphe générique (corps - propriétés)

```
//INDIVIDUALS
72      OPTIONAL {"
73              SELECT ?i ?iL WHERE {"
74                      ?i rdf:type CLASSE.
76                      ?i rdfs:label ?iLpl .
77                      FILTER (langMatches(lang(?iLpl),LANGTAG))
78                      BIND(STRDT(STR(?iLpl), xsd:string) AS ?iL) "
79              }
80      }
        //MAPPING INDIVIDUAL-PROPERTIES
81      OPTIONAL { "
82              ?i ?s ?i2 . "
84              OPTIONAL { "
85                      ?i2 rdfs:label ?lailp . "
86                      FILTER (langMatches(lang(?lailp),LANGTAG)) "
87                      BIND(STRDT(STR(?lailp), xsd:string) AS ?i2L) "
88              }
89      }
90}
```

Requête 5.11 Construction du graphe générique (corps - individus)

Dans les Requête 5.8 à Requête 5.11, les mots en caractères gras sont des variables. Voici leur définition

- **CLASSE** est une chaîne de caractères contenant l'URI de la classe que l'on désire ramener;
- **PREDPONDLENGTH** est un nombre entier correspondant au nombre de classes entre la classe pivot et la classe recherchée;
- **PATHQUERY** est une liste de triplets contenant le chemin des prédicats entre la classe pivot et la classe recherchée. Celui-ci prend la forme de :
  - CLASSE om:path_0 ex:PREDONT_0;
  - CLASSE om:path_1 ex:PREDONT_1;
  - Etc.;

  Où ex:PREDPONT_X sont les prédicats faisant le lien entre les classes;
- **PATHRANGEQUERY** est une liste de triplets contenant le chemin des codomaines entre la classe pivot et la classe recherchée. Celui-ci prend la forme de :
  - CLASSE om:pathRange_0 ex:CODOMAINE_0;
  - CLASSE om:pathRange_1 ex:CODOMAINE_1;
  - Etc.;

  Où ex:CODOMAINE_X sont les codomaines des prédicats faisant le lien entre les classes;
- **LANGTAG** est une chaîne de caractères correspondant à la langue recherchée.

Cette requête construit donc un graphe de solution qui est presque un sous-ensemble du graphe original. Les différences par rapport au graphe original sont les suivantes :

1) Son arrangement par classe, propriétés et individus qui permettent d'accéder rapidement à l'un ou l'autre lors de la programmation des composants Web (lignes 2 à 5);
2) Les triplets contenus dans PATHQUERY et PATHRANGEQUERY qui permettent de fabriquer rapidement les chemins servant à identifier uniquement les différentes manifestations d'une même classe dans la grille sémantique. Ces triplets ne sont pas contenus dans le graphe original;
3) Les triplets formés grâce aux variables ?val et ?val2 (lignes 19 et 20) qui sont des artifices de programmation utiles à la génération automatique du contexte du JSON-LD par Jena.

> Lorsqu'une propriété n'étant jamais évaluée est ramenée, comme celle-ci n'est jamais utilisée comme prédicat mais uniquement comme sujet ou objet, elle n'est pas ajoutée automatiquement au contexte du graphe par Jena. En liant ces variables de façon différente lorsqu'il s'agit d'une propriété objet ou d'une propriété de données, on obtient une utilisation par défaut des propriétés et le contexte est alors généré correctement (lignes 66-67). Ces artifices sont nécessaires du fait de la vision OO de l'outil Web.

Le reste des triplets du sous-graphe sont des triplets du graphe contenu dans la BDT, après inférence.

Les Requête 5.8 à Requête 5.11 ramènent des triplets relatifs à la classe **CLASSE** (lignes 26 à 40), des triplets relatifs aux propriétés ?s dont le domaine ou le domaine directe sont la **CLASSE** (lignes 41 à 71) et des triplets relatifs aux individus ?i du type de cette **CLASSE** (lignes 72 à 80) dont des triplets contenant les valeurs des propriétés ?s pour ces individus ?i (lignes 81 à 89).

Comme on peut s'en douter, cette requête avait un temps de réponse prohibitif quand effectuée contre le graphe complet dans la BDT. Cependant, effectuée en mémoire et sur un sous-graphe, elle s'effectue très rapidement.

Une limitation des Requête 5.8 à Requête 5.11 telles qu'elles sont présentées, est la nécessité que chaque classe, propriété et individu possède une et une seule étiquette et un maximum d'un commentaire pour une langue donnée. Chacune de ces étiquettes et chacun de ces commentaires sont affublées de leur type de données (*xsd:string*) pour faciliter leur utilisation dans l'interface Web (lignes 28-29, 33-34, 51-52, 56-57, 77-78, 86-87). Des stratégies devraient donc être implémentées pour accommoder des ontologies ne répondant pas à ce critère. Pour l'instant une seule étiquette et un seul commentaire sont ramené par la requête et leur choix est fait de manière aléatoire par la BDT.

Les Requête 5.8 à Requête 5.11, ramènent également les types des propriétés qui permettront de déterminer si elles sont fonctionnelles et si elles sont de type objets ou de type données (ligne 60). Elle ramène en outre leur propriété inverse si elle existe (ligne 66) et leur codomaine direct (ligne 59).

En ce qui concerne les individus, lorsqu'on ramène les triplets contenant à la fois leur URI et les URI des propriétés, on doit garder en tête que les objets de tels triplets peuvent être les individus d'une autre classe. C'est pourquoi on ramène également les étiquettes de ces individus car elles seront affichées dans la grille (lignes 84 à 88).

Les Requête 5.8 à Requête 5.11 ramènent aussi toutes les propriétés d'annotation de l'OV caractérisant la classe ou les propriétés (lignes 38 et 63). C'est ici que la Requête 5.7 est utile afin de pouvoir laisser la liberté à l'application d'évoluer. Lors de l'ajout de fonctionnalités à l'application, celles-ci doivent souvent pouvoir faire appel à de nouvelles sémantiques de l'OV, puisque c'est grâce à elles qu'on pourra annoter les modèles de données et ainsi diriger l'utilisation des données à partir de l'ontologie. La Requête 5.7 vient donc ramener systématiquement toutes les propriétés de l'OV et les lignes 38 et 63 peuvent alors trouver tous les triplets contenant des sémantiques de l'OV caractérisant respectivement la classe ou les propriétés désirées. En effet, toutes les propriétés de l'OV sont des sous-propriétés de super-propriétés spécialement pour effectuer ce genre de requête. De cette façon, l'ajout d'une nouvelle propriété à l'OV n'entraîne aucun besoin de modifier les requêtes génériques.

```
1 CONSTRUCT {
2         uap:reponse uap:visualisation ?v .
3         uap:reponse rdfs:label ?lbl .
4         uap:reponse uap:simple ?simple .
5         uap:reponse uap:vueURI <vueURI> .
6         ?v ?v2 ?v3 .
7         ?v3 ?v4 ?v5 .
8         ?v5 ?v6 ?v7 .
9  }
10 WHERE {"
11        OPTIONAL { "
12                <vueURI> uap:visualisation ?v .
13                <vueURI> rdfs:label ?lbl .
14                OPTIONAL { <vueURI> uap:simple ?simple . }
15                OPTIONAL {
16                        ?v2 rdfs:subPropertyOf uap:propDeViz .
17                        ?v ?v2 ?v3 .
18                        OPTIONAL {
19                                ?v4 rdfs:subPropertyOf uap:propDeViz .
20                                ?v3 ?v4 ?v5 .
21                                OPTIONAL {
22                                        ?v6 rdfs:subPropertyOf uap:propDeViz .
23                                        ?v5 ?v6 ?v7 .
24                                }
25                        }
26                }
27        }
28 }
```

Requête 5.12 Retrouver les informations d'une sauvegarde

Dans le même ordre d'idée, dans le cas où l'on ramène la classe pivot d'une sauvegarde, on veut aussi ramener les informations de visualisation contenues dans la sauvegarde. Celles-ci se retrouvent grâce à la Requête 5.12.

Comme toutes les propriétés de sauvegarde de l'OV sont déclarées comme étant des sous-propriétés de *om:propDeViz*, cette requête permet de ramener toutes les informations de visualisation, peu importe la profondeur du patron qu'elles nécessitent, sans ramener des informations superflues. De plus cette façon de faire, permet de faire évoluer l'OV sans avoir à modifier cette requête. Ce sous-graphe est ensuite ajouté au graphe de la classe pivot.

Toutes ces requêtes, sauf la Requête 5.12, sont effectuées autant de fois qu'il y a de classe dans la grille sémantique. Les graphes qui en résultent sont classés dans un tableau de graphe en JSON-LD qui est envoyé à l'interface client qui en fera des JSON classés dans des pages de magasin de données d'Ext.js, avant d'être utilisés pour fabriquer la grille.

5.2.11.2 Services pour les opérations CRUD

Toutes les opérations CRUD s'effectuent de la même manière. L'interface Web récolte les valeurs qui ont changées dans les formulaires, effectue leur mise en forme et envoie deux graphes au serveur : le graphe des valeurs à ajouter et le graphe des valeurs à enlever.

Le serveur va ensuite itérer sur chacun de ces graphes pour créer deux requêtes SPARQL, une des triplets à enlever et une des triplets à ajouter.

Lors de l'ajout de triplets, deux vérifications doivent être effectuées pour assurer la validité logique du graphe résultant. On doit tout d'abord vérifier qu'une propriété fonctionnelle ne sera pas multivaluée par l'opération. On doit ensuite vérifier que la propriété inverse du prédicat du triplet, si elle existe et qu'elle est fonctionnelle, ne soit pas non plus multivaluée par l'opération.

Puisque l'interface Web empêche déjà qu'une propriété fonctionnelle soit multivaluée, si le triplet ne passe pas le test de la multivaluation, c'est que deux usagers sont en train de modifier

le triplet simultanément, alors on doit les en notifier. Si la vérification de la multivaluation inverse n'est pas passée, on en avertit l'usager en lui signifiant la nature du problème.

Lorsqu'on enlève des triplets, on doit vérifier que celui-ci existe bel et bien avant de le supprimer. Si ce n'est pas le cas, c'est qu'un autre usager l'a modifié entre temps, on en notifie l'utilisateur. Si le triplet existe toujours, on le supprime et on doit aussi supprimer les triplets déduits des propriétés inverses, si celles-ci existent.

Lorsqu'on veut effacer un individu complètement de la base de connaissances, on procède alors par l'effacement de tous les triplets contenant l'URI de cet individu comme sujet, comme prédicat ou comme objet. Cela assure que toutes les informations concernant cet individu sont complètement éradiquées de la BDT.

5.2.11.3 Autres services

En plus des services présentés précédemment d'autres services ont été mis en place. Ceux-ci servent à peupler l'arbre et les menus déroulants, enregistrer les sauvegardes, effectuer les inférences et générer des URI.

Les services ce construction de l'arbre sont très simples et consistent à ramener la liste des classes du modèle. Pour chaque classe on ramènera son étiquette (*rdfs:label*) et les commentaires (*rdfs:comment*) s'il y en a. La liste des usagers ayant créé des vues sauvegardées grâce à OM est aussi ramenée afin de présenter les sauvegardes par ballot d'usagers dans l'arbre du mode complet.

Les menus déroulants des différents composants Web quant à eux ne peuvent pas nécessairement être peuplés à partir des informations déjà ramenées du côté Web par les véhicules génériques. C'est pourquoi un service indépendant se charge de ramener les individus d'une classe et leur étiquette lorsque nécessaire.

En ce qui concerne les sauvegardes, dans l'interface Web, chaque action de l'usager modifiant l'état de la grille ou les composantes est enregistrée dans un magasin de données de l'état de la grille. Lorsque l'usager sauvegarde une vue, ce magasin est transformé en JSON puis en JSON-LD auquel sont ajoutées les informations entrées dans le formulaire de sauvegarde, le

nom de l'usager, la date et l'heure. Un nouvel URI est créé pour la sauvegarde et de celui-ci est déduit d'autres URI servant aux éléments de la sauvegarde ayant plusieurs niveaux de profondeurs. Le graphe de la sauvegarde est ensuite enregistré dans la BDT et devient accessible dans la liste des classes et dans la liste des actions de l'interface Web.

Tel que mentionné précédemment, le temps pour générer les inférences est prohibitif et elle ne sont donc pas effectuées à chaque changement fait par les utilisateurs dans la BDT. C'est pourquoi un service indépendant permet aux super-utilisateurs de lancer l'inférence sur demande. Les ensembles de règles d'inférence OWLPRIME et RDFS sont alors exécutés sur le modèle et les données. De plus, ce service est appelé automatiquement tous les jours en période creuse.

Lors de l'ajout d'individus ou de vues, afin de s'assurer de générer des URI uniques peu importe l'ontologie et le jeu de données utilisés, un service permet leur création grâce aux *Globally Unique Identifier* (GUID). Ces identifiants de 16 octets, comporte 122 bits aléatoires et deux GUID ont alors une chance sur $5x10^{36}$ de collision. Les GUID ainsi créés sont concaténés à l'URL d'OM afin de générer de nouveaux URI.

5.2.12 Ontologie de visualisation

L'OV permet de dynamiser l'affichage des composantes en fonction des annotations qu'elle permet de faire au modèle de données. Elle contient aussi les sémantiques décrivant les sauvegardes effectuées avec OM. Cette ontologie ne contient aucune classe, ni aucun individu, mais uniquement des propriétés. Celles-ci sont de quatre types : les propriétés d'annotation de classe, les propriétés d'annotation de propriété, les propriétés d'annotation de sauvegarde et les propriétés d'annotation d'individu.

Les propriétés d'annotation de l'OV peuvent servir à indiquer qu'une classe ou qu'une propriété peut être utilisée avec certaines fonctionnalités d'OM. On pourra par exemple indiquer que les étiquettes des individus d'une classe peuvent ou doivent être générées automatiquement en fonction de la valeur de certaines propriétés.

Elles peuvent aussi servir à encadrer les actions de l'usager. Par exemple, elles pourront indiquer les minima et maxima des propriétés numériques ou stipuler que les individus d'une classe doivent posséder une combinaison de valeur unique à certaines propriétés.

Enfin, elles pourront servir à donner plus d'information à l'utilisateur sur le modèle, par exemple en indiquant l'unité d'une propriété numérique.

Dans une approche comme OM, afin de conserver la généricité des applications, toutes les informations utiles à la dynamisation des composantes de l'interface devraient idéalement se trouver à l'intérieur du modèle de données. L'OV a permis de rendre certaines fonctionnalités d'OM génériques aux ontologies annotées avec l'OV.

L'OV contient aussi des propriétés d'annotation des sauvegardes permettant de persister les informations nécessaires au chargement d'une vue, puisque dans OM, les sauvegardes se font sous forme de graphe RDF dans la BDT.

L'OV contient en outre des propriétés s'appliquant sur tous les individus de la BDT. Ce ne sont pas des propriétés d'annotation à proprement parler comme les autres propriétés de l'OV, puisqu'au sens de RDFS une propriété d'annotation ne participe pas à l'inférence. Ce sont des propriétés de type données afin que, par inférence, elles soient associées à toutes les classes du modèle. Elles ont été présentées brièvement précédemment.

Ces propriétés servent à assurer un suivi sur les modifications faites à la BDT. Tout individu ajouté à la BDT reçoit automatiquement une valeur pour chacune de ces propriétés afin d'indiquer quel usager a créé ou modifié l'individu et quand. Ces propriétés permettent d'assurer une certaine qualité aux données en implémentant un mécanisme d'imputabilité de la validité de celles-ci. De plus, ces propriétés permettent l'implémentation d'une permission sur les vues qui ne permet qu'au créateur d'un individu de pouvoir modifier ses valeurs.

Les propriétés d'annotations de l'ontologie de visualisation sont toutes déclarées comme étant des sous-propriétés des trois grands groupes d'annotations, soit *om:propDeClasse*, *om:propDeProp* et *om:propDeViz* afin de faciliter leur récupération de manière générique.

L'ontologie de visualisation est présentée dans sa totalité dans l'ANNEXE III.

5.3 Conclusion

Dans ce chapitre a été présenté OM, un cadre de développement pour générer des applications auto-adaptives, ainsi que PPAL, la première application générée grâce à celui-ci.

OM est à la fois un constructeur de vue et un navigateur sémantique en fonction qu'on l'utilise respectivement en mode complet ou en mode simple, ce dernier n'offrant qu'un sous-ensemble des fonctionnalités du mode complet afin de protéger les utilisateurs néophytes de la complexité de l'outil.

Comme le CRV-HQ, la première phase d'exploration ontologique commence par l'utilisation d'un arbre présentant les classes de l'ontologie. OM offre aux utilisateurs avancés deux paradigmes d'interactions distincts pour la deuxième phase d'exploration de l'ontologie de domaine : la grille sémantique qui montre en temps réel l'espace de solution et l'onglet de requête graphique qui présente visuellement la requête formée implicitement. Ces deux outils se modifient réciproquement par l'utilisation de l'un ou de l'autre.

Plusieurs mécanismes et fonctionnalités appuient la grille sémantique. Une pléthore d'outils sur les lignes, sur les colonnes et sur les cellules permettent aux utilisateurs de créer toutes les projections possibles d'un graphe multidimensionnel dans une matrice en deux dimensions en plus de permettre de modifier le contenu de la BDT.

Pour supporter la modification de la vue, des filtres permettent de restreindre l'espace de solution et ceux-ci se manœuvrent à partir de composants Web s'adaptant aux codomaines et aux types des propriétés de l'ontologie. Deux saveurs de filtrages des valeurs nulles permettent de nuancer ces filtres. De plus, l'OV peut être utilisée pour annoter l'ontologie et ainsi personnaliser davantage les filtres.

La modification des individus se fait à travers deux types de formulaires semblables générés dynamiquement et se basant sur les codomaines et les types des propriétés pour offrir différentes possibilités d'interactions. Ces formulaires génèrent deux graphes RDF de modifications à appliquer sur la BDT : un graphe des triplets à supprimer et un graphe des triplets à ajouter. Ces deux graphes permettent de couvrir simultanément toutes les opérations CRUD pouvant être effectuées.

La grille sémantique permet la visualisation constante de l'ensemble des résultats grâce à des requêtes génériques qui interrogent la BDT classe par classe. Les résultats de chaque requête peuvent être réconciliés dans la grille grâce aux URI des individus, aux URI des propriétés et aux chemins des prédicats.

Chaque action de l'usager sur la grille entraîne des écritures dans une table de l'état de la grille qui est utilisée pour sauvegarder et charger les vues. Ces sauvegardes se font sous le format RDF en utilisant les propriétés de l'OV.

OM applique sur les données toutes les inférences de RDFS et OWLPRIME. Celles-ci sont appliquées *a priori* sur l'agrégation du TBox et de l'ABox et réappliquées à chaque jour en période creuse. Oracle ne permettant pas le chaîne arrière, des artifices ont dû être développés pour assurer la cohérence logique des données lors des opérations de modification ou de suppression de triplets par les utilisateurs.

Tout comme pour le CRV-HQ, les paradigmes de classification et d'héritage ont dû être réconciliés afin de pouvoir retrouver génériquement les propriétés des classes et de leurs super-classes sans passer par les individus. Cela s'est fait par la création d'un ensemble de pré-inférences qui doit être appliqué sur le TBox avant l'utilisation des jeux RDFS et OWL.

Le côté serveur d'OM est léger et n'est qu'un lieu transitoire où ne sont pas persistées les classes ni les propriétés de l'ontologie afin de garantir l'auto-adaptivité de l'outil. On y retrouve, entre autres, des services pour construire l'arbre, construire la grille, sauvegarder les vues et effectuer les opérations CRUD. Les requêtes génériques permettant de peupler la grille ont été présentées. Celles-ci sont le cœur d'OM, assurant à la fois que l'outil fonctionne avec n'importe quelle ontologie OWL mais aussi que ces ontologies puissent évoluer sans que l'application Web nécessite de modification ou de recompilation.

OM fait usage d'une OV servant à annoter les ontologies de domaines afin de guider les fonctionnalités des composants Web. Elle est utilisée pour annoter les classes, les propriétés et les individus, mais aussi pour permettre les sauvegardes et pour implémenter un mécanisme d'imputabilité de la véracité des données.

Au chapitre suivant, les fonctionnalités de l'outil seront comparées à d'autres outils similaires et à des cadres théoriques. Les enjeux techniques et les réflexions théoriques découlant de son

développement seront abordés. De plus, les nombreuses leçons apprises seront présentées et, parmi celles-ci, les obligations et les recommandations de modélisation ontologique pour optimiser l'utilisation de l'approche permettront de juger de la généralité et de l'universalité de celle-ci.

CHAPITRE 6

DISCUSSION

Dans DATE, une première ébauche d'un cadre de développement d'applications auto-adaptatives a été utilisée pour créer une application client-serveur d'aide à la décision. En utilisant les services génériques du cadre, les classes et les propriétés du modèle conceptuel peuvent être modifiées directement dans la BDT sans impliquer la modification de l'application. La présentation du UI s'ajuste alors automatiquement selon la plus récente mise à jour du modèle puisque les services interrogent systématiquement ce modèle à chaque requête. Cette façon de faire diffère des applications conventionnelles où le modèle des données n'est pris en considération qu'au moment de la compilation. Le cadre proposé supporte toutes les opérations CRUD sur les individus et différentes techniques de visualisation des résultats (grilles et graphiques).

Cette première application était une preuve de concept montrant comment on peut atteindre l'indépendance conceptuelle à travers diverses fonctionnalités. Les applications subséquentes ont montré l'évolution des bases jetées lors du développement de DATE.

Le CRV-HQ a été construit sur les apprentissages faits pendant la réalisation de DATE. Il a cependant porté l'implémentation des technologies du WS à un autre niveau. L'exploration du modèle est devenue possible grâce à des mécanismes génériques, l'expressivité de SPARQL a été presque complètement harnachée et le nombre de fonctionnalités a bondi. Le CRV-HQ a permis d'étudier les mécanismes sous-tendant un outil sémantique Web fortement expressif.

Finalement, OM réutilise des fonctionnalités et des apprentissages à la fois de DATE et du CRV-HQ, dans une application d'un nouveau genre, plus achevée et offrant un meilleur compromis entre l'expressivité et l'utilisabilité. OM est un constructeur de vues tablant sur les propriétés des technologies du WS pour permettre d'une part l'implémentation de nouveaux paradigmes de visualisation tels que la grille sémantique qui est au centre de l'application. Le cadre de développement d'OM sert d'autre part à protéger les utilisateurs de la complexité des technologies du WS en offrant une interface manœuvrable par des néophytes de ces

technologies. Par rapport au CRV-HQ, cette complexité est donc déplacée du UI vers ses composants Web et les services qui les soutiennent.

Cette complexité pourrait être perçue comme un frein à l'évolution du cadre et des applications construites avec celui-ci. Cependant, puisque les applications ainsi construites sont auto-adaptatives, elles peuvent évoluer avec un minimum de modification à leur code.

Plusieurs enjeux doivent être pris en considération lorsqu'on tente d'évaluer des applications telles que DATE, le CRV-HQ et OM. Tout d'abord, cette discussion portera sur les fonctionnalités de ces trois outils. Celles-ci seront comparées à celles d'outils similaires et à des cadres théoriques. Par la suite, des enjeux techniques et des réflexions théoriques ayant émanés de leur développement seront abordés. Finalement, les leçons apprises à chaque étape de cette recherche seront présentées.

6.1 Comparaison des fonctionnalités

Tel que vu au CHAPITRE 1, la littérature abonde d'autres applications sémantiques et de cadres théoriques pour tenter de les classifier et de les évaluer. Les fonctionnalités de DATE, du CRV-HQ et d'OM seront maintenant comparées avec celles de plusieurs autres applications de leur type et avec les fonctionnalités, exigences et lignes directrices issues d'études phares du monde sémantique.

6.1.1 DATE

DATE peut être considéré comme un navigateur textuel pour utilisateur débutant. Ses fonctionnalités seront donc maintenant comparées tout d'abord à d'autres navigateurs textuels, puis à un cadre théorique des fonctionnalités que devraient possédées les applications sémantiques pour utilisateurs débutants.

6.1.1.1 Fonctionnalités des navigateurs textuels

Des fonctionnalités d'autres navigateurs textuels sont présentées dans l'étude de Marie et Gandon (2014). Outre les fonctionnalités de base qu'on peut s'attendre de n'importe quel navigateur sémantique, i.e., de pouvoir naviguer entre les différentes ressources et d'afficher des informations complémentaires sur celles-ci, Marie et Gandon présentent aussi les fonctionnalités uniques de quelques applications.

Ils y parlent entre autres de Noadster (Noadster, 2020) un navigateur qui performe un clustering des données en se basant sur les propriétés et utilisent les clusters pour montrer les propriétés les plus fréquemment utilisées en premier dans un arbre des résultats (Figure 6.1). La possibilité de pouvoir effectuer un clustering sur les données afin d'ordonner les ressources de l'arbre n'est pas utile dans le cas d'utilisation de DATE. Mais dans des applications plus complexes, telles que le CRV-HQ ou OM, cette fonctionnalité pourrait permettre d'assister l'utilisateur à débuter son exploration ontologique. Tel que mentionné dans l'étude de Heim et Ziegler (2009), des fonctionnalités pour guider l'utilisateur à l'initiation de la tâche sont essentielles afin d'harnacher le plein potentiel d'un outil d'exploration sémantique.

Marie et Gandon parlent aussi de Disco (Disco, 2020) un navigateur textuel qui montre les données RDF dans des colonnes de paires propriété-valeur associées à la ressource explorée (Figure 6.2). DATE fait aussi cette association mais va plus loin en le faisant sous forme de grille, ce qui permet toute sorte d'action sur la grille pour construire l'espace de solution désiré. Ces actions seront particulièrement importantes dans le CRV-HQ et OM.

Finalement, Marbles (Marbles, 2020) organise les triplets RDF en fonction de données additionnelles sur la ressource explorée, lesquelles proviennent d'index, de recherches et de revues de systèmes sémantiques. Des bulles de couleurs aident l'utilisateur à identifier la provenance des données (Figure 6.3). Quoi que la provenance de ces informations ne soit pas la même dans le cas de DATE, l'idée reste la même, soit l'utilisation d'une ontologie de visualisation pour guider les composants du UI dans la présentation des données. En ce qui a trait à la provenance, Marbles l'utilise uniquement pour assister l'utilisateur dans sa compréhension de la requête alors que DATE s'en sert aussi pour guider le UI lors des opérations CRUD.

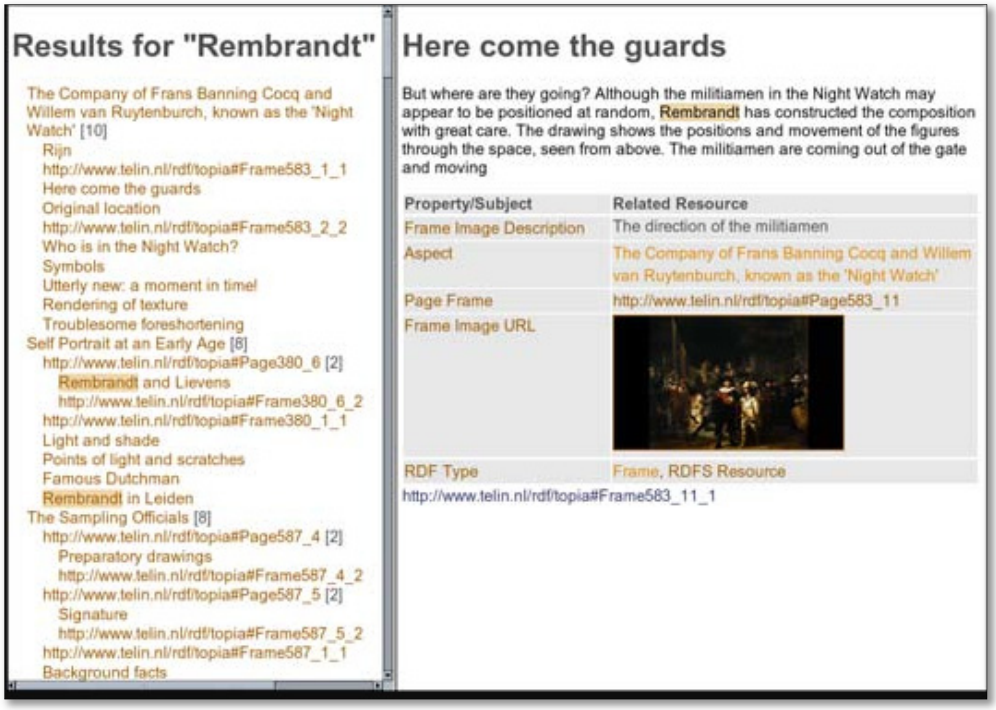

Figure 6.1 Noadster

Source : Tiré de Rutledge, Van Ossenbruggen, & Hardman (2004)

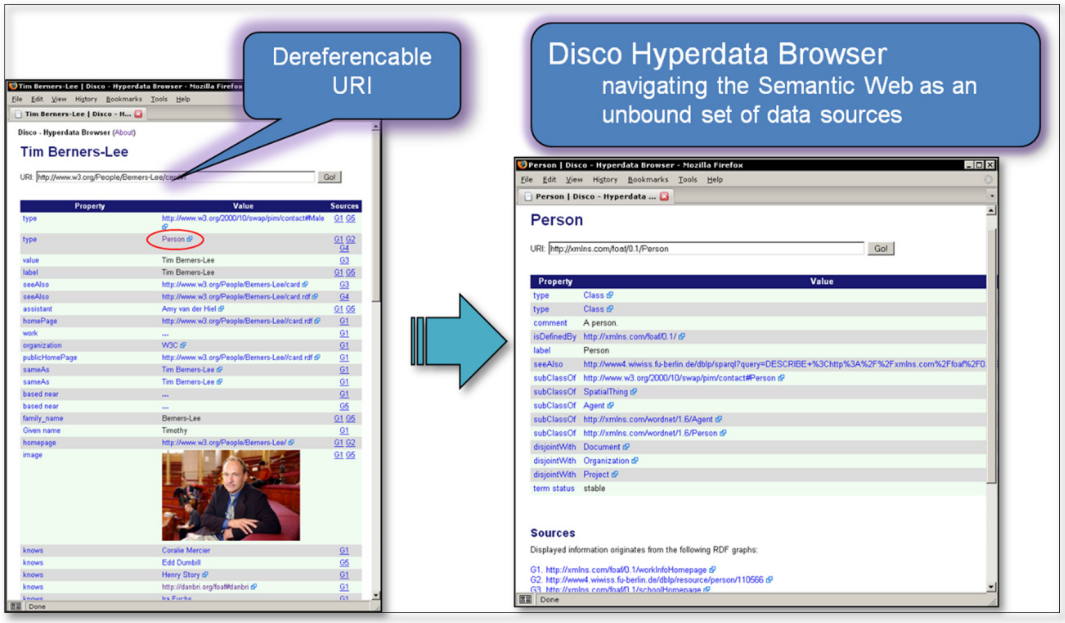

Figure 6.2 Disco

Source : Tiré de Disco (2020)

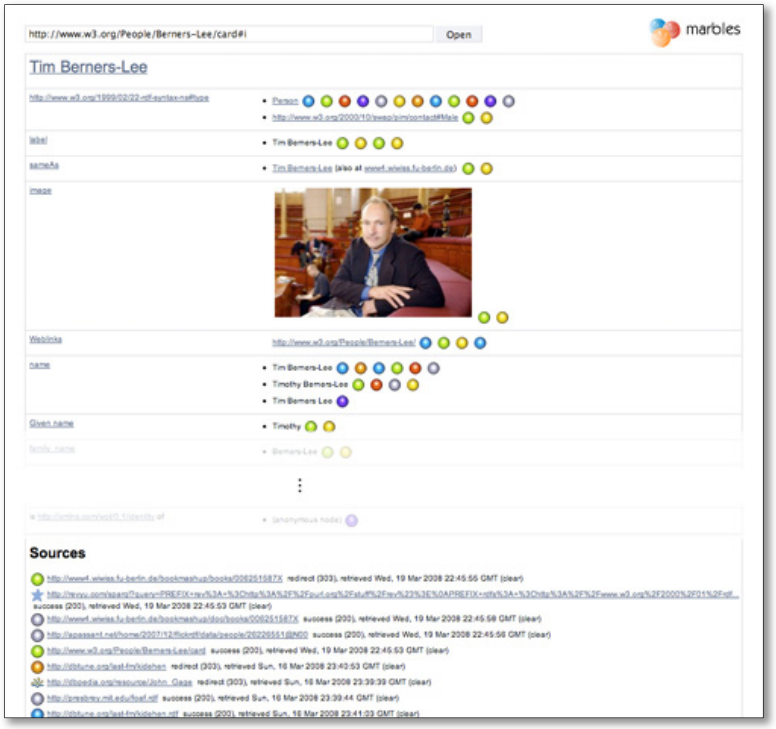

Figure 6.3 Marbles

Source : Tiré de Marbles (2020)

Ces fonctionnalités sont reproduites dans le Tableau 6.1 ci-dessous qui indique aussi si ces fonctionnalités sont présentes dans DATE et l'effort qu'il faudrait déployer pour les y implémenter. La liste est loin d'être exhaustive, mais est constituée des fonctionnalités saillantes permettant de différencier divers navigateurs textuels.

Tableau 6.1 Présence des fonctionnalités des navigateurs textuels dans DATE

Navigateur	**Fonctionnalités des navigateurs textuels**	**Présence**	**Effort**
Noadster	Clusterise les données pour ordonner les ressources dans l'arbre	Non	Grand
Disco	Montre les résultats dans des colonnes de paires propriété-valeur	Oui	
Marbles	Formatte les triplets en fonction de données additionnelles	En partie	Moyen à grand
	Utilise la provenance des données	Oui	

6.1.1.2 Fonctionnalités des applications sémantiques pour utilisateur débutant

DATE a été développée pour être utilisée par des utilisateurs novices. Les recommandations de Dadzie et Rowe (2011) en ce qui a trait au développement d'une application consommant des données liées pour ce type d'utilisateur sont résumées dans le Tableau 6.2.

DATE fournie une navigation intuitive à travers un graphe de données multidimensionnelles. Cette navigation se résume à parcourir les transformateurs contenus dans des postes spécifiques, ou à parcourir les transformateurs contenus dans des postes en fonction des dates des prises de mesures. Elle ne permet pas une exploration *ad hoc* des données.

Cette application ne supporte que des requêtes simples de récupération d'information. En récupérant toutes les classes pouvant donner des détails sur un transformateur sélectionné, on peut considérer qu'elle offre une analyse des régions d'intérêts. Il est également possible d'ajouter des données dans la base de connaissances et donc de les publier. Les résultats des requêtes sont présentés dans des grilles qui peuvent être triées et ordonnées et l'information peut aussi être présentée sous forme de graphiques.

Tableau 6.2 Présence des fonctionnalités des applications sémantiques pour utilisateur débutant dans DATE

Fonctionnalités des applications sémantiques pour utilisateur débutant	**Présence**	**Effort**
Navigation intuitive à travers un large graphe de données complexes et multidimensionnelles	Limitée	Grand
Possibilités de recherche exploratoire des connaissances	Limitée	Grand
Support pour des requêtes simples et avancées (filtrage et récupération d'information),	Limitée	Grand
Analyse détaillée des régions d'intérêts	Limitée	Moyen
Publication des données	Oui	
Extraction de données	Non	Petit
Présentation des résultats des analyses	Oui	

Outre l'extraction des données, DATE présente donc toutes les fonctionnalités attendues d'un navigateur sémantique pour utilisateur débutant, même si certaines de celles-ci sont encore très sommaires. DATE se démarque des autres navigateurs textuels en se spécialisant dans la

présentation de données scientifiques sous forme de table et en présentant des rudiments d'auto-adaptabilité.

6.1.2 CRV-HQ

Le CRV-HQ est un CRV sémantique destiné autant aux utilisateurs débutants qu'aux utilisateurs avancés. Ses fonctionnalités ont été comparées avec celles du CRV relationnel Toad, celles de plusieurs CRV sémantiques, ainsi qu'avec celles prescrites par des cadres théoriques pour tous types d'applications utilisant les données liées. De plus, ses fonctionnalités ont aussi été comparées aux fonctionnalités attendues à la fois des applications sémantiques pour utilisateurs débutants et pour utilisateurs avancés.

6.1.2.1 Fonctionnalités du CRV de Toad

Le fonctionnement de base du CRV-HQ ressemble beaucoup au CRV de Toad, ce dernier offrant toutefois beaucoup plus de fonctionnalités et de possibilités d'interactions. Dans le CRV de Toad, l'usager glisse et dépose des tables, vues ou synonymes depuis le navigateur de schéma, le gestionnaire de projet, la palette d'objets ou la fenêtre de recherche d'objets. Le navigateur de schéma étant l'équivalent de la liste des classes du CRV-HQ, les deux fonctionnent de manières sensiblement similaires. Dans le CRV de Toad l'usager peut toutefois choisir plus d'une table depuis le navigateur de schéma. Le soin d'effectuer les jointures entre les classes lui étant laissé, la forme de sa requête en sera plus libre. Cependant, les connaissances requises pour pouvoir fabriquer des requêtes ayant du sens sont corrélées proportionnellement à cette liberté.

Ensuite, l'utilisateur doit sélectionner les colonnes des tables qu'il désire faire entrer dans la composition de sa requête. Ceci s'effectue en explorant la liste des attributs disponibles apparaissant dans les boîtes représentant ces tables et en cochant la case à cocher appropriée. Cette façon de procéder est donc similaire pour les deux CRV.

Dans le CRV de Toad, lorsqu'une table est introduite dans l'espace de travail, si elle possède des jointures possibles avec d'autres tables, ces jointures s'affichent automatiquement. Dans le CRV-HQ, cela se traduit par les propriétés de type objets qui se trouvent dans la liste des propriétés de la classe.

Le CRV de Toad permet à l'usager de glisser et déposer des colonnes d'une table sur une autre table pour créer des jointures. Le clic droit sur une jointure permet de spécifier le type de jointures ainsi que d'ajouter des conditions sur la jointure (=, >, <, ...). Encore une fois, cette façon de procéder ne limite pas l'usager dans la forme des requêtes. Cependant, en présence d'un très grand schéma, l'exploration des possibilités de requêtes peut devenir fastidieuse. Ces fonctionnalités ne peuvent pas se retrouver exactement telles quelles dans le CRV-HQ dû à l'utilisation d'une classe pivot et la nécessité de créer des requêtes en arbres. Cette façon de faire est utilisée par plusieurs CRV SPARQL et sera discutée plus en détails plus loin. Cela dit, les conditions sur les jointures ne se retrouvent pas telles quelles dans SPARQL, mais des conditions sur les classes permettent d'arriver à un résultat similaire.

L'usager du CRV de Toad pourra utiliser l'outil de création et d'édition de BDR de Toad pour visualiser le modèle et réfléchir sur la formulation de sa requête. Un tel outil montrant l'ensemble de l'ontologie serait très intéressant à implémenter dans le CRV-HQ. Cependant, il faudra tout d'abord déterminer comment représenter visuellement plusieurs sémantiques OWL, ce qui ne représente pas une tâche triviale. C'est en fait un domaine d'étude en soi, dont on ne citera ici que Héon, Nkambou, & Gaha (2016) dont les recherches ont eu lieu à l'IREQ.

À chaque fois qu'une colonne de table est sélectionnée pour faire partie d'une requête, elle s'ajoute à la grille des colonnes. Cette grille permet alors d'ajouter des clauses d'agrégation, des clauses WHERE, GROUP BY, HAVING ou ORDER BY, des clauses de tri, de rendre les colonnes invisibles dans les résultats et de donner des alias aux champs ou aux tables. Ici aussi, l'usager possède une vaste panoplie de possibilités. Le CRV de Toad aide l'usager avec la composition des expressions de ces clauses et le guide en fonction des types de données de ces colonnes. Le CRV-HQ présente les filtres d'une manière similaire dans une vue en formulaire permettant d'ajouter des clauses sur les propriétés. Des outils pour guider l'utilisateur dans les

choix des filtres, comme les statistiques sur les modalités, sont aussi implémentés sous une forme plus simple.

Le CRV de Toad permet les requêtes imbriquées (*subqueries*) et l'union de requêtes (*union queries*). De plus, il permet les sous-requêtes de types EXISTS ou NOT EXISTS. L'arborescence de ces sous-requêtes et unions est représentée dans la vue en arbre de la requête. Le CRV-HQ ne permet pas de faire de sous-requêtes, mais leur implémentation serait possible. Les clauses EXISTS et NOT EXISTS faisant partie des fonctionnalités de SPARQL, leur implémentation serait aussi possible et simple à réaliser.

Le CRV de Toad offre l'option d'améliorer une requête, soit en utilisant le SQL Optimizer, un outil de Toad World ou en utilisant le Oracle Tuning Advisor, un outil d'Oracle. Ces deux outils permettent d'identifier automatiquement les problèmes de performance potentiels des requêtes SQL et d'explorer les options d'optimisation. Malheureusement, à notre connaissance il n'existe pas de tels outils pour les requêtes SPARQL effectuées sur Oracle. L'optimisation des requêtes est un grand enjeu pour ce genre d'outil et il sera discuté plus loin.

Le CRV de Toad permet d'écrire une requête manuellement. Le CRV-HQ le fait aussi, cependant, l'espace graphique ne s'ajuste pas en conséquence comme le fait Toad.

Finalement, Toad for Oracle permet de réaménager automatiquement les tables dans l'espace de travail, de changer leur taille automatiquement, de créer et d'éditer des champs de calculs, d'avoir des détails supplémentaires sur les tables, de visualiser l'impact des changements dans le diagramme sur la requête SQL grâce à un code de couleur de surlignement, de publier les requêtes, d'envoyer les requêtes à un module d'automatisation et d'exporter les requêtes vers Microsoft Excel (Excel, 2020). Toutes ces fonctionnalités sont possibles à implémenter dans le CRV-HQ mais seulement les possibilités de publication des requêtes et l'exportation des requêtes vers Excel (par l'utilisation d'un fichier CSV) sont présentes dans la version actuelle.

Le Tableau 6.3 récapitule les propriétés qui ont été observées dans le CRV de Toad. On y voit lesquelles sont actuellement présentes dans le CRV-HQ et, si ce n'est pas le cas, une estimation de l'effort nécessaire pour arriver à les implémenter.

Tableau 6.3 Présence des fonctionnalités du CRV de Toad dans le CRV-HQ

Fonctionnalités du CRV de Toad	**Présence**	**Effort**
Glisser et déposer des classes dans l'espace graphique	Limitée (1 table)	Très grand
Glisser et déposer des vues dans l'espace graphique	Non	Moyen à grand
Rechercher des objets conceptuels dans l'ontologie	Oui	
Lister les colonnes des tables (propriétés de données)	Oui	
Sélectionner des propriétés des classes désirées dans la requête	Oui	
Lister les jointures possibles (propriétés objets)	Oui	
Spécifier le type de jointure	Oui (OPTIONAL)	
Mettre des conditions sur les jointures	Oui, par les filtres	
Accéder à un outil de création et d'édition de modèle	Non	Grand
Accéder à une vue complète du modèle	Non	Petit à Moyen
Offrir une vue en formulaire des actions possibles sur les colonnes	Oui	
Effectuer des tris (WHERE)	Oui	
Utiliser l'agrégation (GROUP BY, HAVING)	Oui	
Ordonner les résultats (ORDER BY)	Oui	
Rendre une colonne invisible dans les résultats	Oui	
Donner des alias aux champs ou aux tables	Non	Petit
Aider l'usager avec la composition des expressions des clauses	Limitée	Petit à grand
Effectuer des requêtes imbriquées	Non	Moyen
Effectuer l'union de requêtes	Non	Moyen
Créer des sous-requêtes de types EXISTS ou NOT EXISTS	Non	Petit
Optimiser automatiquement les requêtes	Non	Très grand
Écrire des requêtes manuellement	Oui	
Ajuster l'espace graphique selon les modifications manuelles	Non	Grand
Réaménager automatiquement l'espace de travail	Non	Très petit
Redimensionner automatiquement les classes	Non	Très petit
Créer et éditer des champs de calculs	Non	Moyen à grand
Avoir des détails supplémentaires sur des tables	Oui	
Visualiser le nombre des valeurs impliquées dans les jointures	Non	Très petit
Publier les requêtes	Oui	
Envoyer les requêtes à un module d'automatisation	Non	Moyen à grand
Exporter les requêtes vers Microsoft Excel	Oui	
Visualiser en temps réel la requête textuelle créée	Non	Très grand
Visualiser l'impact des changements grâce à un code de couleur	Non	Petit

6.1.2.2 Fonctionnalités d'autres CRV SPARQL

Les fonctionnalités du CRV-HQ ont été comparées aux fonctionnalités de Facet Graphs (gFacet), OptiqueVQS, Mash-QL, SparqlFilterFlow, GRQL et SPARQLGraph.

Facet Graphs (Heim & Ziegler, 2009; Heim, Ertl, & Ziegler, 2010), est probablement le CRV SPARQL ressemblant le plus visuellement au CRV-HQ (Figure 6.4). Tout comme le CRV de Toad, il utilise une approche en graphe où les nœuds représentant les classes sont des boîtes

contenant des listes. Ces boîtes sont reliées entre elles par les relations entre les classes. Une différence majeure entre le CRV-HQ et Facet Graphs vient du fait qu'à l'intérieur des boîtes du second sont présentées les listes des individus de ces classes alors que dans le premier ce sont les listes des propriétés qui y sont présentées. Sélectionner un élément d'une liste de Facet Graphs ne permet pas de naviguer le modèle de données, comme c'est le cas dans le CRV-HQ, mais bien de restreindre l'espace de solution. L'usager y construit donc itérativement l'ensemble des résultats souhaités en sélectionnant des individus des classes comme filtres. En d'autres mots, alors que dans Facet Graphs les facettes sont les membres des classes, dans le CRV-HQ les facettes sont les attributs de ces classes.

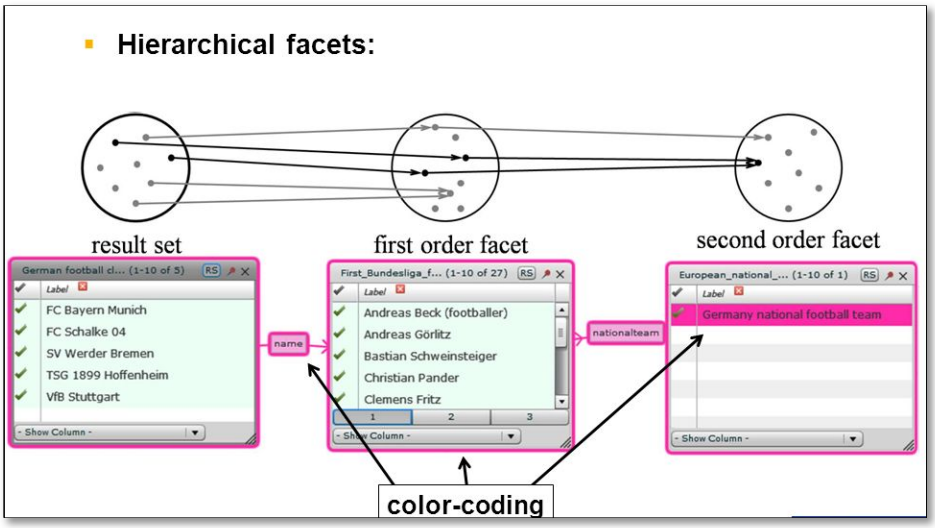

Figure 6.4 Facet Graphs
Source : Tiré de Heim & Ziegler (2009)

Les auteurs affirment que les avantages de représenter les facettes en tant que nœud dans un graphe sont que : les attributs pour chaque facette sont groupés dans un seul nœud, tous les nœuds sont montrés de façon cohérente dans une seule page, les relations sémantiques entre les nœuds sont représentées par des arêtes étiquetées et que les facettes peuvent être ajoutées et enlevées par l'usager. Ces avantages sont autant valables pour le CRV-HQ, malgré la différente nature des facettes.

Il pourrait être envisageable d'offrir aux usagers la possibilité de choisir entre deux modes du CRV-HQ, un utilisant les propriétés comme facettes et l'autre utilisant les individus comme facettes. Cela dit, l'effort demandé serait grand pour une augmentation d'utilisabilité qui ne semble pas garantie.

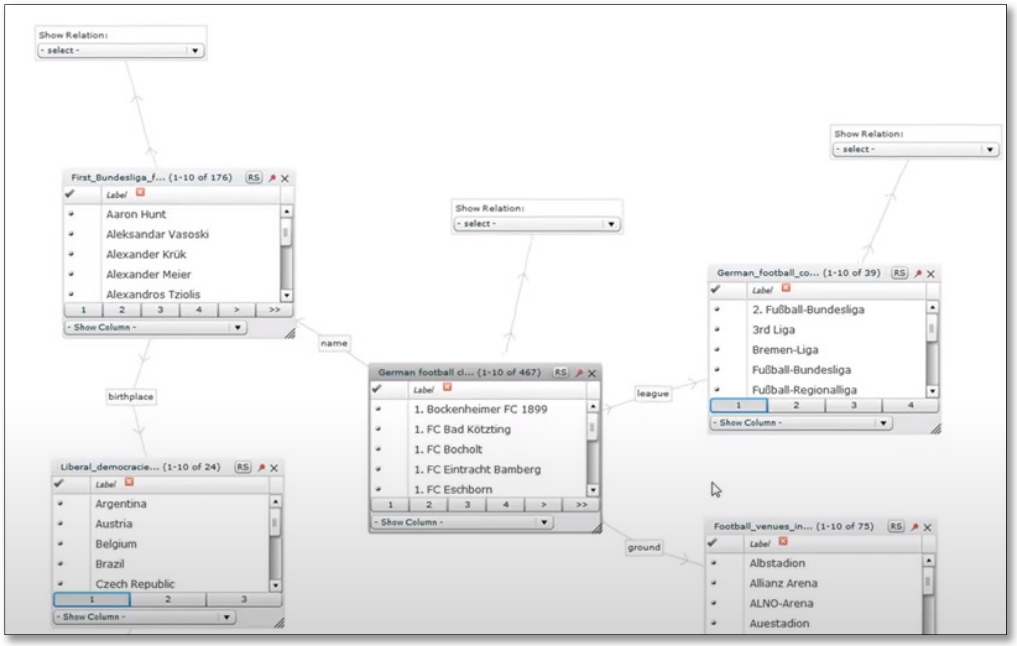

Figure 6.5 gFacet
Source : Tiré de Heim, Ertl, & Ziegler (2010)

Dans Facet Graphs, à chaque fois qu'un attribut d'une facette est sélectionné, toutes les autres facettes sont mises à jour pour représenter l'espace de solution résultant (Figure 6.5). De cette façon, l'usager n'arrive jamais à une requête sans résultat. Dans des versions subséquentes du CRV-HQ, cette idée devrait être exploitée. Les créateurs d'Optique VQS dont il sera question plus loin, discutent aussi de telles possibilités de rétroaction dans Soylu et Giese (2016).

Mash-QL est un langage de formulation de requêtes menant à un CRV SPARQL dont plusieurs articles ont été publiés depuis 2008 (Jarrar & Dikaiakos, 2008; Jarrar & Dikaiakos, 2008; Jarrar & Dikaiakos, 2009; Jarrar & Dikaiakos, 2011; Medhe & Phalke, 2012). On y retrouve plusieurs

des éléments principaux qui composent des outils comme Optique VQS ou le CRV-HQ (Figure 6.6).

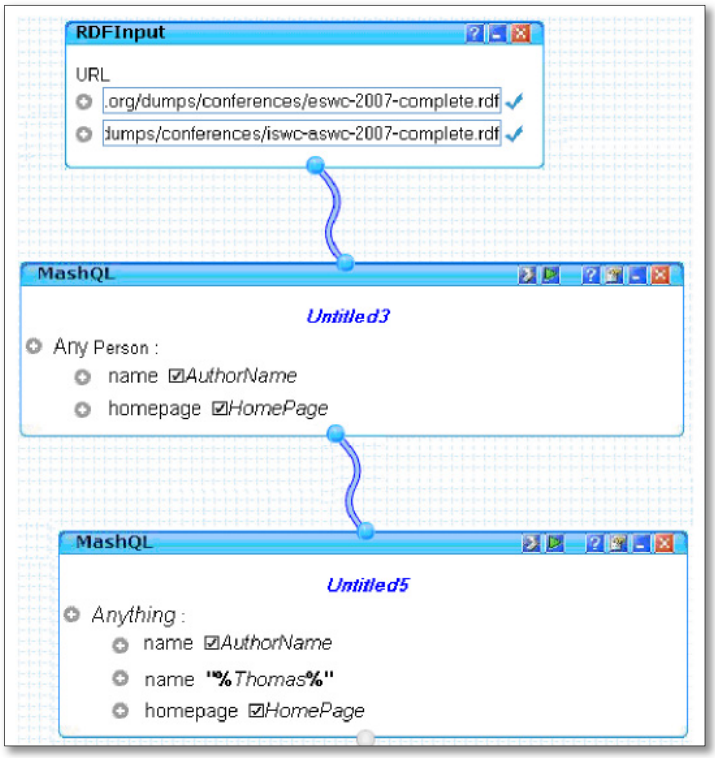

Figure 6.6 MashQL

Source : Tiré de Jarrar & Dikaiakos (2008)

Le but de leur outil est de permettre une sérialisation en XML des requêtes visuelles créées par les usagers afin que ces requêtes soient facilement enregistrées, chargées, exécutées et partagées (Jarrar & Dikaiakos, 2008). Les principales contributions de leur outil sont de protéger l'usager de la complexité de SPARQL, en rendant les variables et l'espace des noms (*namespace*) implicites donc transparents à l'utilisateur, de permettre les requêtes sur plusieurs sources de données qui peuvent ou non avoir de schéma de données, tout en étant suffisamment expressif (Jarrar & Dikaiakos, 2011). Ils sont possiblement les premiers à utiliser une structure de requête en arbre, simplifiant ainsi la traduction de la requête en langage SPARQL (Jarrar & Dikaiakos, 2008).

Le fait de pouvoir explorer des données dont le schéma n'est pas connu d'avance entre dans le paradigme de l'auto-adaptivité des systèmes d'information, tel qu'étudié dans le cadre de DATE et d'OM. En outre, leur implémentation sépare la requête en trois parties, la tête (*Header*), le corps (*Body*) et le pied (*Footer*), ce qui permet à leur engin de construire itérativement les requêtes.

Ces éléments se retrouvent dans le CRV-HQ avec quelques différences. Premièrement, la sérialisation du CRV-HQ est faite en format RDF au lieu de XML, ce qui permet de stocker les requêtes dans une BDT et ce qui permettra à terme d'analyser facilement le jeu de requêtes en fonction de l'ontologie questionnée. Deuxièmement, leur approche visuelle de choisir toutes les propriétés de la requête par des menus déroulants temporaires (contrairement aux listes de facettes qui restent accessibles en permanence à l'usager dans le CRV-HQ) semble mal indiquée pour une ontologie de la taille de l'IEC/CIM et plus complexe à utiliser pour les usagers. Troisièmement, Mash-QL étant construit pour explorer des données possiblement sans schéma ou provenant de schémas différents, les noms des propriétés sont déduits à partir des URI. Le CRV-HQ est quant à lui conçu pour explorer des ontologies formelles. Les noms des propriétés et des classes sont alors contenus sous la propriété d'annotation *rdfs:label*, ce qui rend l'implémentation plus simple et évite les collisions de noms par sa conception même.

Dans une version ultérieure du CRV-HQ, il serait important d'implanter un mécanisme tel que celui contenu dans Mash-QL, pour que l'usager puisse indiquer que deux propriétés différentes issues de deux ontologies distinctes représentent le même concept. De plus, Mash-QL offre la possibilité de lier des requêtes entre elles, de façon à se servir des résultats d'une requête comme intrant d'une autre, à la manière d'un module d'extraction-transformation-chargement (en anglais *Extract*, *Transform*, *Load*, ETL). Cette possibilité est équivalente à l'imbrication de requêtes dans le langage SPARQL et devrait aussi être envisagée dans les prochaines itérations du CRV-HQ.

La performance d'un CRV SPARQL étant un élément crucial à son utilisation, les créateurs de Mash-QL résument très bien la situation vécue pendant la réalisation du CRV-HQ. Le temps que prend l'engin pour traduire la requête visuelle en requête SPARQL est négligeable. La complexité et le temps d'exécution d'une requête visuelle sont donc déterminés par le langage

de requête utilisé (Jarrar & Dikaiakos, 2008) et l'optimisation de son utilisation dans la base de données. Les technologies de BDT et de connections à celles-ci vont donc être les régulateurs de la performance d'un CRV SPARQL et dicter les tailles de graphes maximales ainsi que possibilités de requêtes que celui-ci peut formuler.

Les auteurs spécifient que dans leur cas particulier, c'est le choix des sources qui peuvent être externes et qui doivent être ramenées en mémoire qui peuvent rendre les requêtes intermédiaires sous performantes. Le CRV-HQ utilise aussi des requêtes intermédiaires pour permettre de formuler la requête. Ce genre de problème ne se rencontre pas en pratique puisque ces requêtes intermédiaires sont effectuées sur les schémas et non sur les individus. En revanche, ce genre de problème sera à considérer dans l'application du chapitre suivant où les requêtes intermédiaires impliquent aussi les individus. Jarrar et Dikaiakos (2008) soulignent que les problèmes de performance relèvent de SPARQL lui-même et que ce n'est pas Mash-QL qui est en cause. Cependant, dans le cas du CRV-HQ, il a été réalisé que les optimisateurs de requêtes SPARQL d'Oracle 12c n'étant pas encore optimaux eux-mêmes et que la façon dont sont fabriquées les requêtes peut influencer le temps de calcul. Ceci implique que la traduction de la requête visuelle en requête SPARQL a un impact sur la performance. Une des difficultés pour un CRV SPARQL est donc de choisir une traduction SPARQL parmi un éventail de traductions possibles qui soit la plus performante pour le gestionnaire de triplets choisi.

La pléthore d'articles scientifiques parus dans le cadre du projet européen Optique offre un cadre théorique très bien défini pour les CRV SPARQL (Soylu, *et al.*, 2013; Soylu, *et al.*, 2013; Ahmetaj, Calvanese, Ortiz, & Šimkus, 2014; Bagosi, *et al.*, 2014; Soylu A. , *et al.*, 2014; Giese, *et al.*, 2015; Soylu A. , *et al.*, 2015; Soylu & Giese, 2016). Ce grand projet de 14 millions d'euros financé par la Commission Européenne s'est déroulé sur 4 ans de 2012 à 2016 et a été coordonné par l'université d'Oslo. Ce projet dont le focus portait sur l'utilisation des technologies du WS visait à donner une connexion bout-à-bout entre les utilisateurs et les données, à permettre la formulation intuitive des requêtes, à intégrer de façon transparente les données de plusieurs sources et à exploiter massivement le parallélisme. Il contenait un volet sur le développement d'un CRV SPARQL et c'est dans ce cadre que plusieurs fondements théoriques de ce genre d'outils ont été développés (Figure 6.7).

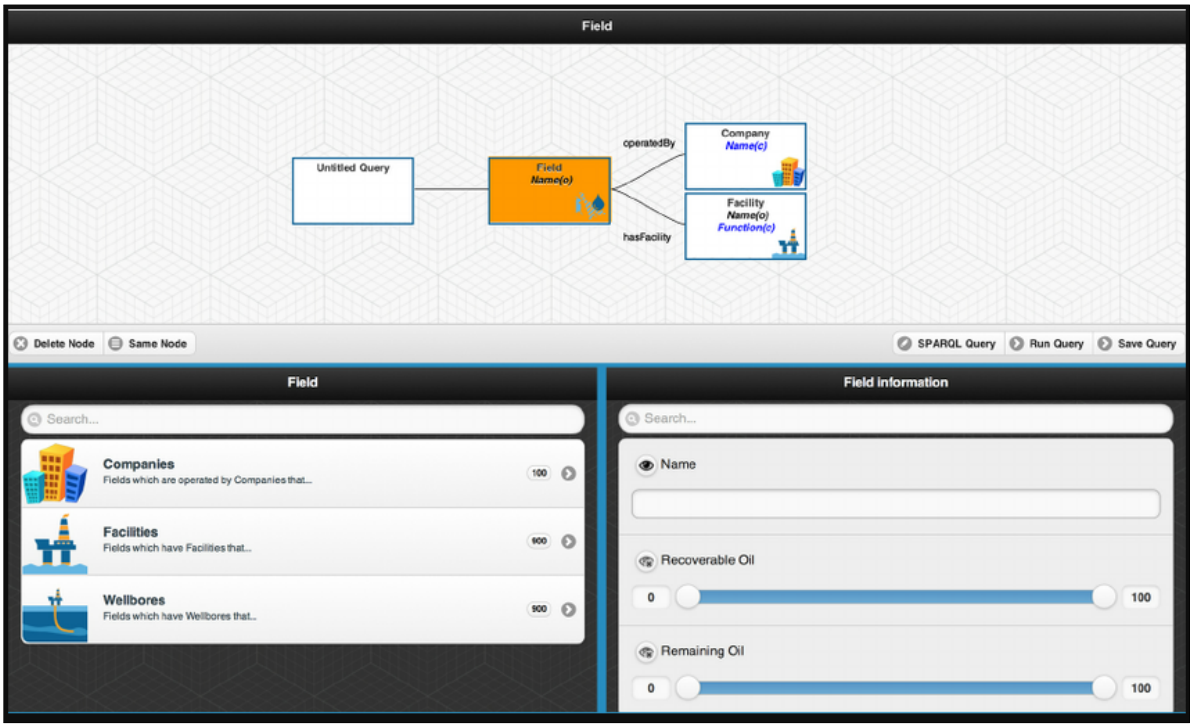

Figure 6.7 Optique VQS
Source : Tiré de Soylu *et al.* (2013)

Optique VQS, même s'il ne ressemble visuellement pas au CRV-HQ, lui est très similaire dans son principe de fonctionnement. Il permet l'exploration des données par leur schéma, ce qui permet, selon eux, à l'utilisateur de mieux comprendre les connaissances du domaine que par une exploration à partir des individus (Soylu, Giese, Jimenez-Ruiz, Vega-Gorgojo, & Horrocks, 2016). Une première fenêtre présente un graphe des classes à l'utilisateur qui peut incrémentalement ajouter d'autres classes grâce aux relations d'association qui sont présentées dans une deuxième fenêtre. Cette deuxième fenêtre présente les relations comme autant de facettes que l'utilisateur peut sélectionner pour explorer l'ontologie (à partir des relations d'association) ou pour appliquer des filtres sur les données (à partir des relations de données) dans une troisième fenêtre prévue à cet effet. Les auteurs utilisent l'idée de construire les requêtes à partir d'une seule classe pivot sélectionnée à partir de l'ensemble des classes de l'ontologie. Cette classe devient le point de départ de la requête et toutes les autres classes qui feront partie de la requête auront un patron ou une suite de patrons lui étant reliés. Bien que les

zones de l'interfaces soient à des endroits différents, ce sont les mêmes paradigmes de visualisation et d'exploration que le CRV-HQ. Une différence notoire est que les listes des propriétés (données et objets) ne sont visibles qu'une classe à la fois, ce qui peut ralentir le temps d'écriture d'une requête.

Tel que mentionné, l'usager navigue dans le graphe pour construire sa requête à partir d'une classe pivot. Cette méthode, qui est aussi utilisée dans le CRV-HQ, Mash-QL (Jarrar & Dikaiakos, 2011), GRQL (Athanasis, Christophides, & Kotzinos, 2004) et Konduit (Ambrus, Möller, & Handschuh, 2010). Elle a été très bien documentée dans le cadre d'Optique. Les auteurs expliquent dans Soylu A., *et al.* (2014) que la requête prend ainsi la forme d'un arbre et l'ajout incrémental de branches vient restreindre l'espace de solution.

Cette façon de faire restreint le nombre de requêtes possibles (Soylu & Giese, 2016) mais a plusieurs avantages. Premièrement, on s'assure qu'aucune boucle infinie ne soit créée, ce qui permet non seulement de faciliter la tâche des usagers mais aussi de pouvoir créer l'engin en utilisant des fonctions récursives. Deuxièmement, comme l'usager n'a plus accès qu'aux classes liées (de près ou de loin) à la première classe sélectionnée, l'espace graphique ne contient jamais de classe dont la relation avec une autre classe n'est pas connue. Cela implique que peu importe les manœuvres de l'usager, les patrons de la requête seront nécessairement liés entre eux, évitant donc par sa conception même les espaces de solution exponentiellement grands qu'entraînent le mélange de patrons non liés au sein d'une même requête, comme cela est permis dans le CRV de Toad. Soylu et Giese (2016) expliquent qu'un CRV SPARQL tel qu'Optique permet de construire des requêtes conjonctives en forme d'arbres en projetant l'ontologie dans une structure de graphe. Ils démontrent ensuite formellement que les requêtes issues de cette tâche non-triviale se conforment au graphe sous-jacent du système.

Lors d'essais effectués dans les laboratoires, Optique VQS ne permettait que de se brancher aux BDT via un point de terminaison SPARQL, ce qui rendait l'exécution plus lente que si l'on avait pu se brancher directement via un connecteur JDBC (en anglais, *Java Database Connectivity*). De plus, l'ontologie des systèmes électriques (IEC/CIM) a semblée trop volumineuse pour être utilisée de façon performante dans Optique. Cela est peut-être dû au fait que des raisonnements sont effectués à chaud sur l'ontologie pour déduire l'héritage des

propriétés. Le CRV-HQ a été conçu pour faire les inférences par lots, sur demande et se branche directement à une BD Oracle via une connexion JDBC, ce qui permet d'obtenir une meilleure performance. Évidemment, le CRV-HQ ne permet pas toutes les possibilités d'interopérabilité que le projet Optique offre, mais son utilisation plus spécialisée est plus performante pour ce cas précis.

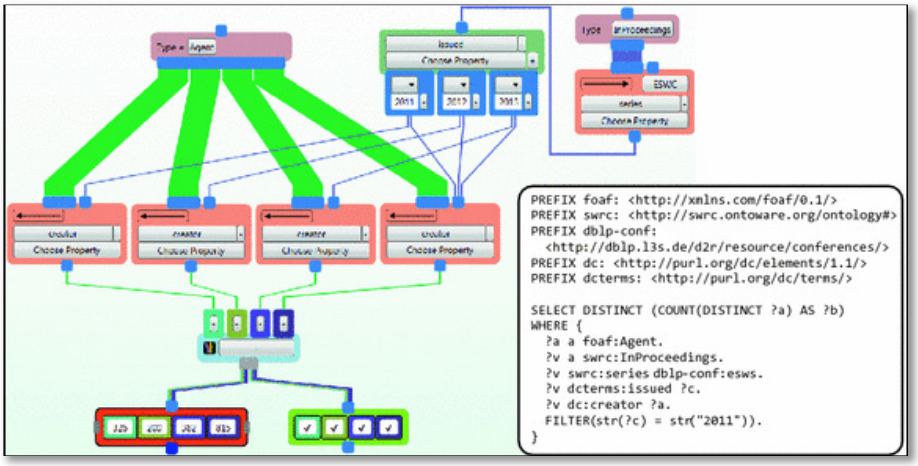

Figure 6.8 SparqlFilterFlow
Source : Tiré de Haag, Lohmann, & Ertl (2014)

SparqlFilterFlow (Haag, Lohmann, & Ertl, 2014; Haag, Lohmann, Bold, & Ertl, 2014), est construit autour de l'idée de représenter des tuyaux visuels (*visual pipes*) combinés avec une approche filtre / écoulement (Figure 6.8). La lourdeur visuelle qui découlerait de l'utilisation des composants visuels de filtres de SparqlFilterFlow dans un outil de l'expressivité du CRV-HQ ne rendrait pas l'approche viable. Cependant, l'idée d'indiquer visuellement le nombre de solutions filtrées par l'ajout d'une classe en faisant varier l'épaisseur des liaisons est très intéressante et pourrait être implantée dans les prochaines versions.

GRQL (Athanasis, Christophides, & Kotzinos, 2004) est un CRV SPARQL dont le principe ressemble également au CRV-HQ. Même si l'interface graphique est très différente (Figure 6.9), les moyens de naviguer l'ontologie restent les mêmes. L'usager parcours l'ontologie grâce aux relations d'association et aux relations de subsomption et peut ajouter des filtres

grâce aux relations de données. Alors que l'engin du CRV-HQ utilise des méthodes récursives pour systématiquement ajouter des patrons aux requêtes SPARQL générées, GRQL utilise un algorithme qui peut venir modifier des patrons déjà ajoutés. C'est une approche plus complexe qui n'apporte pas de nouvelles possibilités de requêtes. GRQL est obligé d'utiliser cette approche puisqu'il traduit les patrons de la requête au fur et à mesure que l'utilisateur construit celle-ci. Cela a l'avantage de permettre d'afficher les résultats de la requête en temps réel. Le CRV-HQ, quant à lui, ne construit les patrons qu'une fois la requête de l'utilisateur terminée, ce qui simplifie l'algorithme de traduction.

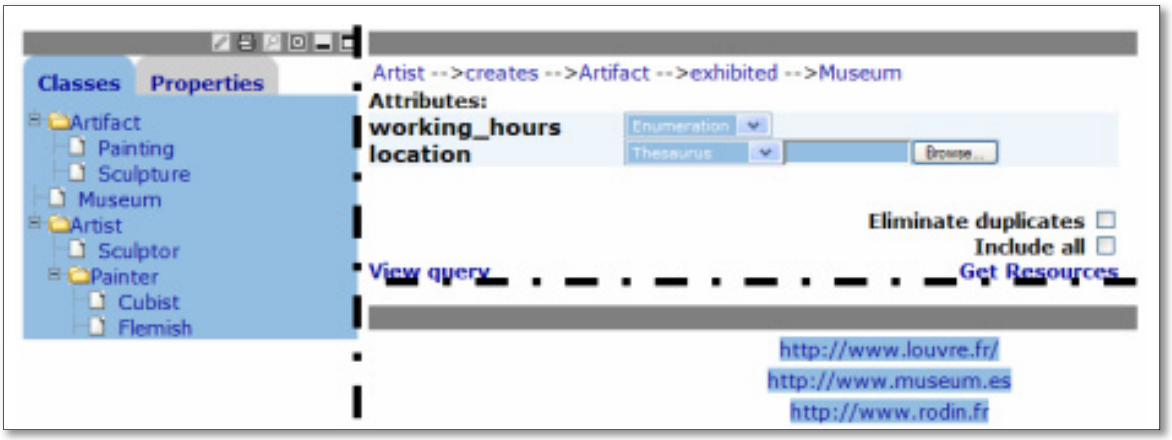

Figure 6.9 GRQL

Source : Tiré de Athanasis, Christophides, & Kotzinos (2004)

GRQL propose aux utilisateurs de choisir une classe mais aussi une relation comme point de départ de la requête, ce que n'offre pas le CRV-HQ. L'utilité d'une telle fonctionnalité pourrait être étudiée, mais il est certain qu'elle apporterait une complexité supplémentaire à l'implémentation.

SPARQLGraph (Schweiger, Trajanoski, & Pabinger, 2014; Schweiger, 2014) est un CRV SPARQL conçu pour fabriquer des requêtes fédérées sur plusieurs grandes bases de données portant sur la chimie et la biologie, rassemblées et mise en format RDF par les projets Bio2RDF (Belleau, Nolin, Tourigny, Rigault, & Morissette, 2008) et EMBL-EBI. Ce CRV incorpore un

système de commentaires intégré permettant aux usagers de discuter et proposer des améliorations ou des modifications sur les requêtes déjà enregistrées par d'autres usagers (Schweiger, Trajanoski, & Pabinger, 2014). Pour faire ces requêtes simultanément sur plusieurs bases de données, SPARQLGraph utilise le mot-clé SERVICE du langage SPARQL. Les auteurs affirment qu'il est le seul CRV SPARQL faisant usage de cette possibilité en 2014 (Figure 6.10).

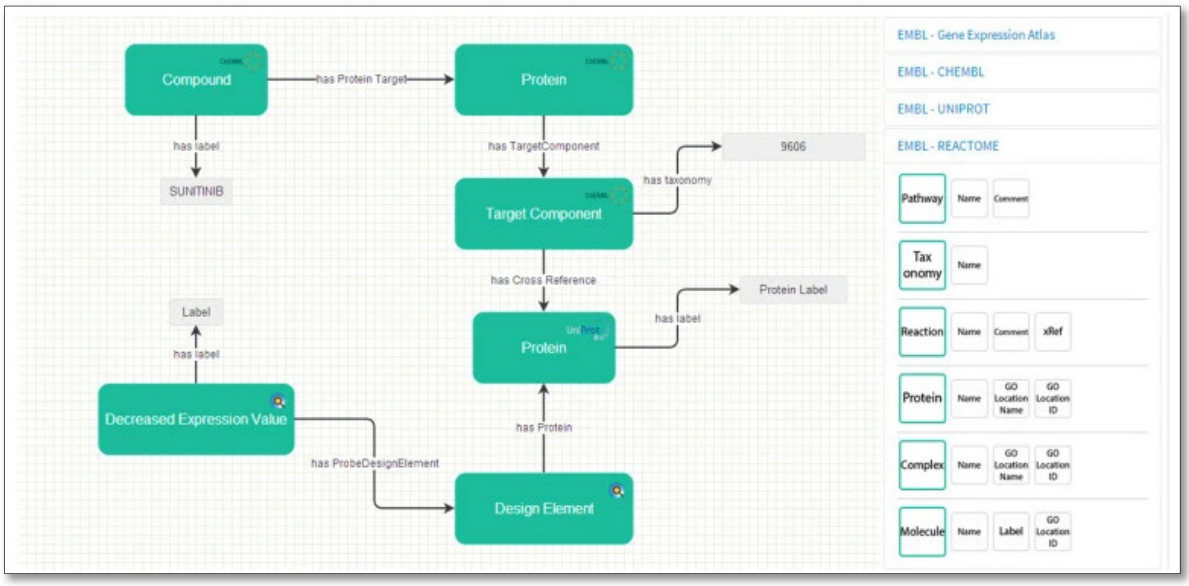

Figure 6.10 SPARQLGraph

Source :Tiré de Schweiger, Trajanoski, & Pabinger (2014)

Ces deux fonctionnalités intéressantes devraient être étudiées dans les versions subséquentes du CRV-HQ. Le créateur de SPARQLGraph indique que son outil est conçu spécialement pour les utilisateurs des domaines de la santé et de la biologie et non de façon générique. SPARQLGraph offre donc la possibilité à l'usager de glisser et déposer seulement certaines classes ou propriétés préalablement sélectionnées dans les bases de données implémentées (Schweiger, 2014). Ces classes et propriétés sont choisies selon leur importance, leur fréquence et leurs possibilités d'interconnexion. Ce travail de sélection doit être fait manuellement.

Ainsi, il incombe à l'usager de relier les classes entre elles en utilisant les bonnes propriétés lorsqu'il construit sa requête visuellement. Il doit donc connaître ou apprendre les sous-ensembles sélectionnés des ontologies pour pouvoir construire ses requêtes efficacement. Des éléments visuels guident l'usager dans ce processus, mais s'il ne connaît pas la structure d'un sous-ensemble, il avancera à tâtons. Cela mène à un outil plus épuré et possiblement plus simple visuellement, mais moins générique et offrant moins de possibilités de requêtes. Cela dit, la spécialisation de l'outil augmente possiblement l'utilisabilité pour les experts du domaine.

Dans SPARQLGraph, les littéraux ne peuvent être filtrés qu'en comparant leur valeur alors que SPARQL offre une panoplie de filtres possibles sur les littéraux. Tout comme le CRV-HQ, des info-bulles permettent à l'usager de mieux comprendre une classe ou une propriété.

Le Tableau 6.4 récapitule les propriétés qui ont été observées dans d'autres CRV SPARQL. On y voit lesquelles de celles-ci sont actuellement présentes dans le CRV-HQ et, sinon, l'effort qu'il faudrait mettre pour y arriver.

Parmi tous ces CRV SPARQL, aucun n'a été conçu en tentant de minimiser l'effort de changement perçu par des usagers habitués aux outils de requêtes visuelles relationnelles, tel qu'il a été fait dans la présente recherche. Un tel outil est pourtant souhaitable car c'est un vecteur d'adoption important de ces nouvelles technologies. La plupart des personnes amenées à interroger les bases de données d'une entreprise ne sont pas intéressées par les technologies impliquées mais par les fonctionnalités qui en découlent.

Camoufler la complexité à l'usager et augmenter les fonctionnalités permettant de générer du savoir à partir des données sont les avantages que tentent d'offrir tous les outils de requêtes visuelles SPARQL. L'outil proposé par cette recherche tente de le faire en réduisant le temps d'adaptation aux nouveaux outils pour les usagers déjà formés aux outils relationnels. Cela représente un gain de productivité pour les entreprises souhaitant adopter les technologies sémantiques en minimisant le temps de formation des employés à ces nouveaux outils en plus de diminuer la résistance des utilisateurs aux changements. De plus, cet outil pourrait permettre, s'il est jumelé à l'outil de génération d'applications sémantiques auto-adaptatives, d'augmenter les possibilités de personnalisation des applications ainsi créées. En outre, sa

composante graphique a permis d'ajouter des mécanismes de visualisation des ontologies à OM, comme il a été présenté au CHAPITRE 5.

Tableau 6.4 Présence des fonctionnalités d'autres CRV SPARQL dans le CRV-HQ

CRV SPARQL	Fonctionnalités présentes dans d'autres CRV SPARQL	Présence	Effort
Facet Graph	Utiliser un graphe où les classes sont des boîtes contenant des listes	Oui	
	Pouvoir utiliser les individus comme facettes	Non	Grand
	Avertir l'usager lorsqu'une requête retournera un ensemble vide	Non	Moyen
Mash-QL	Permettre d'enregistrer, charger, exécuter et partager les requêtes	Oui	
	Protéger l'usager contre la complexité de SPARQL	Limitée	Moyen à Grand
	Permettre des requêtes sur plusieurs sources de données	Non	Petit à Moyen
	Permettre des requêtes sur des données sans schéma	Non	Très grand
	Utiliser une structure de requête en arbre	Oui	
	Choisir les propriétés par des menus déroulants temporaires	Non	Petit
	Déduire les étiquettes à partir des URI	Non	Petit
	Indiquer que deux URI représentent le même concept	Non	Moyen
	Offrir l'imbrication de requêtes	Non	Moyen
	Optimiser les requêtes construites	Non	Grand à Très grand
Optique VQS	Offrir une connexion bout-à-bout entre les utilisateurs et les données	Oui	
	Permettre une formulation intuitive de requête	Oui	
	Intégrer de façon transparente les données de plusieurs sources	Non	Moyen à grand
	Exploiter massivement le parallélisme	Non	Grand
	Explorer les données par leur schéma	Oui	
	Ajouter de façon incrémentale des classes grâce aux relations d'association	Oui	
	Appliquer des filtres grâce aux relations de données	Oui	
	Construire les requêtes à partir d'une seule classe pivot	Oui	
SparqlFilterFlow	Indiquer visuellement l'effet d'un filtre par l'épaisseur des liaisons	Non	Petit
GRQL	Naviguer par les relations de subsomption	Oui	
	Utiliser une relation comme point de départ de la requête	Non	Moyen
SPARQLGraph	Offrir un système de commentaires et discussions sur les requêtes	Non	Moyen
	Utiliser des info-bulles indicatives	Oui	
	Utiliser le mot-clé SERVICE	Non	Moyen

6.1.2.3 Fonctionnalités de l'étude de Heim et Ziegler

Heim et Ziegler (2009) séparent le processus de recherche d'information en six étapes ((1) l'initialisation de la tâche, (2) la sélection, (3) l'exploration, (4) la formulation du focus, (5) la

collecte et (6) la présentation) et décrivent des exigences pour chacune d'elles. Ils comparent ensuite cinq CRV SPARQL selon qu'ils répondent ou non à ces exigences.

Les auteurs suggèrent que, dès l'initialisation de la tâche et tout au long du processus de recherche d'information, un support continuel devrait être fournit à l'utilisateur pour l'aider à définir le problème (1.1) et que des suggestions devraient lui être faites pour l'aider à diriger sa recherche (1.2). Ceci peut être fait en se basant sur les recherches effectuées précédemment par lui ou d'autres usagers et pourrait profiter des techniques de prédiction d'usage issues des recherches en intelligence artificielle. C'est un concept intéressant qui pourrait être envisagé dans le CRV-HQ, mais qui demande un grand effort pour être mis en place.

Lors de la sélection, les auteurs recommandent (2.1) de fournir une vue d'ensemble de tous les sujets (ce qu'offre le CRV-HQ) ainsi (2.2) qu'un système de suggestions automatiques de sujets. Le CRV-HQ n'offre qu'une recherche par mot-clé unique avec auto-complétion, mais des fonctionnalités plus complexes seraient envisageables et souhaitables.

Des cinq exigences de l'étape d'exploration, les quatre premières sont déjà implémentées dans le CRV-HQ, soit (3.1) une représentation graphique intuitive, (3.2) des possibilités d'interactions auto-explicatives et simples d'utilisation, (3.3) des détails sur demande, (3.4) un système de classement et de pagination pour les résultats. La cinquième exigence (3.5) est d'offrir des possibilités de zoomer pour montrer différents niveaux de détails de l'information. C'est une exigence plus compliquée qu'il n'y paraît et un champ d'étude en soi. Une telle fonctionnalité serait en effet très intéressante mais l'effort à fournir serait proportionnel.

L'étape de formulation du focus qui sert à identifier des filtres et les appliquer aux données comprend six exigences, dont trois sont prises en compte par le CRV-HQ, soit (4.1) la formulation et la modification interactive et intuitive des filtres, (4.3) la possibilité de combiner les différents filtres et (4.4) la possibilité de construire des filtres hiérarchiques. Ceux qui ne sont pas encore pris en compte n'en sont pas moins intéressants : (4.2) la présentation immédiate des résultats, (4.5) la traçabilité des effets des filtres et (4.6) la possibilité de changer le focus d'une requête. Ces points devront être considérés dans les versions ultérieures du CRV-HQ. Pour l'instant, la présentation immédiate des résultats entraînerait une baisse de performance qui est non viable dans la forme actuelle de cet outil. C'est en revanche un des

avantages intéressants apporté par l'application OM. La traçabilité des effets des filtres pourrait se faire sans encombrer inutilement la présentation visuelle en ne présentant que la différence du nombre de résultats avant et après, mais cela équivaut à exécuter les requêtes immédiatement à chaque changement. La possibilité de changer de focus est aussi un concept intéressant, mais dont la complexité rend l'implémentation prohibitive due à la conception actuelle de l'engin de traduction des requêtes.

De la collecte, découle les exigences (5.1) d'un mécanisme simple pour sélectionner les découvertes et (5.2) d'exporter les informations sélectionnées dans d'autres systèmes. Le CRV-HQ supporte ces deux exigences.

Finalement, l'étape de présentation comprend la seule exigence (6.1) d'offrir un large éventail de possibilités de visualisation des résultats, ce qui sera assurément pris en compte dans les prochaines versions du CRV-HQ.

Tableau 6.5 Présence des fonctionnalités de l'étude de Heim et Ziegler dans le CRV-HQ

Étape	**Fonctionnalités présentes dans l'étude de Heim et Ziegler**	**Présence**	**Effort**
Initiation de la tâche	R1.1. Un support continuel pour guider l'utilisateur	Non	Très grand
	R1.2. Des suggestions pour guider l'utilisateur	Limitée	Grand
Sélection	R2.1 Une vue d'ensemble de tous les concepts	Limitée	Moyen à grand
	R2.2 Des suggestions automatiques de concepts	Limitée	Moyen à grand
Exploration	R3.1 Une représentation graphique de l'ontologie	Oui	
	R3.2 Des possibilités d'interactions intuitives	Oui	
	R3.3 Des détails sur demande	Oui	
	R3.4 Un classement et une pagination des résultats	Oui	
	R3.5 Une fonctionnalité pour zoomer sur les détails	Non	Grand à très grand
Formulation du focus	R4.1 La formulation interactive et intuitive des filtres	Oui	
	R4.2 La présentation immédiate des résultats	Non	Moyen à grand
	R4.3 La possibilité de combiner les différents filtres	Oui	
	R4.4 La possibilité de construire des filtres hiérarchiques	Oui	
	R4.5 La traçabilité des effets des filtres	Non	Moyen
	R4.6 La possibilité de changer le focus d'une requête	Non	Grand
Collection	R5.1 Un mécanisme facile pour sélectionner les découvertes	Oui	
	R5.2 La possibilité d'exporter les résultats dans d'autres systèmes	Oui	
Présentation	R6.1 Un large éventail de possibilités de visualisation des résultats	Limitée	Petit à très grand

Le Tableau 6.5 récapitule les propriétés qui ont été observées dans Heim & Ziegler (2009). On y voit lesquelles de celles-ci sont actuellement présentes dans le CRV-HQ et, sinon quel effort serait nécessaire pour les implémenter.

6.1.2.4 Exigences, lignes directrices et fonctionnalités de l'étude de Dadzie et Rowe

Dadzie et Rowe (2011) proposent plusieurs cadres pour l'analyse des fonctionnalités des applications. Ils définissent des exigences à lesquelles tout système de visualisation devrait répondre, des exigences à lesquelles une application consommant des données liées destinée aux débutants devrait répondre et des exigences à lesquelles une application consommant des données liées destinée aux experts devrait répondre. De plus, ils définissent des lignes directrices de conception pour ce type d'applications ainsi qu'une série de fonctionnalités qu'elles devraient implémenter.

Les lignes directrices de conceptions et les exigences de tout système de visualisation comportant plusieurs éléments similaires, elles seront traitées simultanément et présentées dans le Tableau 6.6. Dadzie et Rowe, partant du mantra de la visualisation de Shneiderman, expliquent que tout système de visualisation doit d'abord présenter une vue générale, suivie de possibilités de filtres et de sélection et finalement permettre d'obtenir des détails sur demande. La vue générale offerte par le CRV-HQ se trouve dans l'arbre générique et est constituée de la liste des classes regroupées par ballot. Tel que mentionné précédemment, une vue complète de l'ontologie serait intéressante, mais l'implémentation demanderait beaucoup d'efforts. En ce qui a trait aux filtres et aux possibilités de sélection, ils se manœuvrent dans le CRV-HQ par les propriétés (respectivement données et objets) des classes. Les détails sur demande sont implémentés par les bulles contextuelles de commentaires sur les classes et propriétés ainsi que grâce aux exemples de valeurs des données objets. On peut aussi considérer que l'exécution des requêtes est une forme de demande de détails. Dans ce cas, le CRV-HQ remplit très bien cette exigence. Cela dit, des fonctionnalités citées précédemment telles que de pouvoir zoomer sur l'ontologie complète pour voir celle-ci s'afficher sur plusieurs niveaux de détails et d'agrégation permettrait d'encore mieux répondre à ces trois exigences.

Le CRV-HQ permet la manipulation des données multidimensionnelles, des données hiérarchiques et des données de réseaux. Bien que les requêtes ne prennent que la forme d'arbre et que les résultats ne soient que sous forme de matrice en deux dimensions, les données de la BDT sont de toutes ces natures. De plus, il permet principalement d'identifier et de surligner les relations entre les données puisque l'exploration ontologique fonctionne par les propriétés unissant les données aux individus et les individus aux classes. L'extraction de données, elle, se fait par les requêtes qui sont composées pendant cette exploration ontologique.

La conservation d'un historique n'existe que sous la forme de requêtes sauvegardées. Des fonctionnalités de versionnement des requêtes seraient à développer pour améliorer l'outil. Il serait aussi important de pouvoir utiliser les fonctionnalités de retour arrière et de retour avant des navigateurs Web afin de permettre à l'usager de facilement naviguer à travers l'historique de sa navigation ontologique et de sa construction de requête.

Tableau 6.6 Présence des exigences d'un système de visualisation et des lignes directrices de conception dans le CRV-HQ

Exigences de tout système de visualisation	**Présence**	**Effort**
Une vue générale	Limitée	Grand
Des filtres et des possibilités de sélection	Oui	
Des détails sur demande	Oui	
La manipulation de données multidimensionnelles	Oui	
La manipulation de données hiérarchiques (arbres)	Oui	
La manipulation de données de réseaux (graphes)	Oui	
L'identification et le surlignage des relations entre les données	Oui	
L'extraction des données	Oui	
L'historique	Limitée	Moyen

Dadzie et Rowe font la liste d'exigences que les applications sémantiques pour utilisateur débutant devraient implémenter. La navigation intuitive d'un graphe de données, la possibilité de recherche exploratoire des connaissances, le support pour des requêtes simples et avancées, l'analyse détaillée des régions d'intérêt, la publication des données, l'extraction de données et la présentation des résultats des analyses sont toutes couvertes par le CRV-HQ, à différents niveaux.

Parmi les points pouvant être améliorés sans dénaturer l'application, on retrouve le support pour des requêtes simples. La multitude de boutons et de fonctionnalités du CRV-HQ peut être intimidante pour l'utilisateur débutant. Cela dit, s'il ne s'aventure pas dans les fonctionnalités avancées et se contente d'utiliser les propriétés des classes pour créer des patrons de requêtes, l'utilisateur débutant pourra aisément arriver à ses fins.

L'analyse des régions d'intérêts est relativement bien couverte par les commentaires, les exemples de modalités et le résultat des requêtes. La publication des données quant à elle se fait par la publication des sauvegardes. Il n'est cependant pas possible de publier de nouvelles données dans la BDT. L'extraction des données ne peut se faire qu'au format CSV. La présentation des résultats ne se fait pour l'instant que sous forme de tables en deux dimensions et les prochaines versions du CRV-HQ devraient absolument permettre de faire des graphiques à partir de ces résultats. Ces exigences sont récapitulées dans le Tableau 6.7.

Tableau 6.7 Présence des exigences des applications sémantiques pour utilisateur débutant dans le CRV-HQ

Fonctionnalités des applications sémantiques pour utilisateur débutant	**Présence**	**Effort**
Navigation intuitive d'un graphe de données complexes et multidimensionnelles	Oui	
Possibilités de recherche exploratoire des connaissances	Oui	
Support pour des requêtes simples et avancées	Oui	
Analyse détaillée des régions d'intérêts	Oui	
Publication des données	Limitée	Grand
Extraction de données	Limitée	Petit à Moyen
Présentation des résultats des analyses	Limitée	Petit à moyen

En ce qui a trait aux exigences des applications sémantiques pour utilisateurs experts, Dadzie et Rowe citent de nouveau la navigation intuitive des données liées. C'est une exigence qui ne dépend donc pas du type d'utilisateur. Cependant, les utilisateurs experts pourraient désirer des fonctionnalités plus avancées dans cette navigation. Le CRV-HQ permet une navigation ontologique plus intuitive pour les experts de domaine, ceux-ci ayant une connaissance plus avancée des concepts contenus dans l'ontologie.

L'exploration des données permettant de saisir leur structure et leur contenu et pour identifier les liens à travers le graphe semble aussi adéquate pour ce type d'utilisateur. Le CRV-HQ n'a pas été conçu pour identifier les erreurs et les anomalies. Certaines de celles-ci pourront être trouvées par l'utilisation judicieuse de filtres dans les requêtes, mais d'autres, telles que la présence d'îlots de données, ne seront pas détectables par l'outil. De même, la vérification et la validation des données pourront être faites par la fabrication de requêtes à cette fin, mais l'outil ne prévoit pas de fonctionnalités dédiées à ce but.

Le support pour des requêtes avancées utilisant une syntaxe formelle est très bien établi et la possibilité de modifier manuellement les requêtes vient surenchérir sur cette exigence. La publication des données à l'aide des sauvegardes de requêtes qui sont partagées entre les utilisateurs est aussi adéquate. En revanche, l'extraction des données est limitée au format CSV. D'autres formats, tels que RDF seraient probablement appréciés des utilisateurs experts.

Ces exigences sont résumées dans le Tableau 6.8, qui montre aussi leur présence dans le CRV-HQ et les efforts qu'il faudrait mettre pour implémenter celles qui ne le sont pas.

Tableau 6.8 Présence des exigences des applications sémantiques pour utilisateur expert dans le CRV-HQ

Fonctionnalités des applications sémantiques pour utilisateur expert	**Présence**	**Effort**
La navigation intuitive dans les données liées	Oui	
L'exploration des données permettant de saisir leur structure et leur contenu	Oui	
L'exploration des données pour identifier les liens à travers le graphe	Oui	
L'exploration des données pour identifier les erreurs et les anomalies	Non	Moyen à grand
Le support pour des requêtes avancées utilisant une syntaxe formelle	Oui	
La publication des données	Limitée	Grand
La vérification et la validation des données	Non	Moyen à grand
L'extraction des données pour qu'elles puissent être utilisées dans d'autres systèmes	Limitée	Petit à Moyen

Finalement, Dadzie et Rowe énumèrent les fonctionnalités que les applications de données liées peuvent présentées. Celles-ci se retrouvent dans le Tableau 6.9.

Les canevas de présentation permettant d'associer les types de données à des standards de présentation se retrouvent à même les composants de l'outil Web. En effet, ceux-ci vont utiliser les types des données afin de modifier leurs comportements. Comme il a été mentionné dans

DATE et dans OM, le développement d'une ontologie de présentation peut jouer ce rôle. Dans le CRV-HQ, seulement quelques annotations sont faites au modèle de données pour augmenter les fonctionnalités, mais les développements ultérieurs en nécessiteront probablement davantage.

Le point d'entrée dans les données du CRV-HQ est la liste des classes. On entre donc dans les données par le modèle. Encore une fois, l'affichage d'une visualisation graphique de l'ontologie complète pourrait fournir un meilleur moyen de sélectionner le point d'entrée. De plus, des fonctionnalités de recherche par mots-clés pourraient aussi permettre de trouver automatiquement la ou les classes dont les propriétés ou les données contiennent une certaine valeur ou une certaine chaîne de caractères.

La généricité de l'outil lui permettant de fonctionner avec des ontologies de différents domaines est présente ainsi que la recherche par facettes. Cet outil ne permet cependant pas la publication de nouvelles données ou l'édition des données existantes dans la BDT.

Tableau 6.9 Présence des fonctionnalités liées aux applications de données liées dans le CRV-HQ

Fonctionnalités des applications de données liées	**Présence**	**Effort**
Des canevas de présentation permettant d'associer les types de données à des standards de présentation	Oui	
Des points d'entrée dans les données liées, par une URI ou par des mots-clés ou encore grâce à une carte	Limitée	Moyen à très grand
Une généricité permettant à l'outil de parcourir des données de différents domaines	Oui	
La recherche et navigation par facettes pour supporter une recherche intuitive de données multidimensionnelles et mettre en évidence des relations cachées	Oui	
La publication de nouvelles données	Limitée	Grand
La fusion de différents jeux de données	Non	Petit
L'indication de la provenance des données et de la confiance qu'on peut avoir en elles	Non	Petit à Moyen
L'édition des données pour pouvoir les enrichir et les corriger	Non	Grand
La réutilisabilité des extrants	Limitée	Petit à Moyen

La fusion de différents jeux de données est une fonctionnalité qui est absente de l'outil. Son implémentation pourrait se faire facilement. L'indication de la provenance des données et de la confiance qu'on peut avoir en elles fait aussi défaut, mais des mécanismes d'imputabilité de

la véracité des données comme ceux développés dans OM (CHAPITRE 5) pourraient être ajoutés au CRV-HQ. Finalement, la réutilisabilité des extrants est assurée par l'export des résultats vers le format CSV et l'offre d'autres formats d'exportation serait souhaitable.

6.1.2.5 Retour sur les comparaisons

Les sept tableaux précédents démontrent que le CRV-HQ pourrait éventuellement offrir toutes les fonctionnalités des autres CRV analysés, SQL ou SPARQL, et qu'il pourrait aussi répondre aux exigences prescrites par les différents cadres théoriques. Des pistes pour combler chacune des lacunes ont été identifiées et l'effort pour y arriver a été estimé.

6.1.3 OM

OM peut être considéré comme un générateur d'applications sémantiques auto-adaptatives manœuvrables par des utilisateurs débutants. L'approche d'OM a donc été comparée avec les approches d'autres générateurs d'applications sémantiques. Ses fonctionnalités ont été comparées à celles recommandées pour tous les outils pour données liées ainsi qu'aux exigences et lignes directrices que devraient possédées à la fois les outils pour utilisateurs débutants et pour utilisateurs experts.

6.1.3.1 Approche

L'approche d'OM met entre les mains des super-utilisateurs le pouvoir de créer les vues qu'ils désirent par l'exploration *ad hoc* des données du domaine.

Une telle approche semble pouvoir réduire le temps investit par les experts informatiques dans le développement et la maintenance d'applications en diminuant le temps pour déterminer les besoins fonctionnels des clients et en laissant le soin aux experts des domaines de créer et modifier les vues de l'application.

Cette approche est différente d'une approche comme celle de SWP qui laisse le soin aux ontologistes de définir des relations complexes et flexibles sur les données en plus de dicter comment une donnée doit être représentée à une interface utilisateur. Quoiqu'elle aide à enrichir les ontologies et clarifier la représentation qui devrait en être faite, c'est une approche d'implémentation complexe. Elle implique qu'un usager ne connaissant pas les technologies sémantiques ne pourra pas facilement utiliser un tel langage ni modifier les applications à sa guise. De plus, il n'est pas clair si cette approche est propice à une exploration intuitive des modèles de données. Dans l'approche proposée par la présente recherche, les composantes génériques s'adaptent aux standards les plus utilisés dans les modèles sémantiques et ainsi n'entraînent pas la nécessité de modifier les techniques actuelles de développement ontologiques pour être utilisées. De plus, cette approche permet à l'utilisateur final d'adapter la vue sur les données, ce qui ne semble pas facilement possible dans une approche comme celle de SWP.

Contrairement à la majorité des solutions existantes pour la génération automatisée d'applications, la méthodologie qui est proposée par la présente recherche permet la création d'applications sans avoir à écrire de code. C'est l'implémentation des propriétés d'auto-adaptativité au modèle de données qui permet à un canevas d'application d'être réutilisé pour créer de multiples applications.

À notre connaissance, aucune recherche ne tente de jumeler des composantes génériques auto-adaptatives à une construction d'application dirigée par une ontologie à l'exception notable d'inova8 (inova8, 2020) qui présente aussi une approche sans génération de code. Cependant, alors qu'inova8 utilise des canevas d'applications qui peuvent être édités par un programmeur, la solution proposée ici met l'édition dans les mains de l'utilisateur grâce aux composants Web.

De tous les outils de visualisation analysés dans la présente recherche aucun ne permet de construire une vue dynamiquement par l'exploration des facettes que sont les propriétés dans une seule et même grille. L'avantage de cette grille est double; non seulement elle permet d'explorer le modèle de données en ayant la vue résultante constamment sous les yeux, sa nature générique lui permet de créer des vues sur n'importe quelle ontologie. Des mécanismes

d'enregistrement des vues sont utilisés avec celle-ci et permettent ainsi la création d'applications sémantiques auto-adaptatives.

De plus, aucune des recherches analysées ne se concentre sur la création de tels outils pour des utilisateurs non experts en informatique et néophytes des technologies du WS. Ces éléments jumelés permettent de croire qu'OM est le premier outil d'aide à la génération d'applications auto-adaptatives guidées par les ontologies se servant de composantes génériques pouvant être personnalisées et manœuvrées par des utilisateurs débutants.

6.1.3.2 Fonctionnalités d'autres outils pour données liées

Les fonctionnalités de plusieurs outils de visualisation de données Web ont été étudiées dans le cadre de la présente recherche. Marie et Gandon (2014), ont fait un excellent travail de recensement des fonctionnalités originales qui distinguent de tels outils.

RDF Gravity (2020) permet le filtrage, la recherche et le repositionnement des nœuds dans une interface permettant de visualiser un graphe RDF ou OWL (voir Figure 6.11). Le type des ressources est traduit en couleurs pour aider la compréhension. Ces mêmes fonctionnalités sont présentes dans OM, mais implémentées bien différemment. Alors que RDF Gravity représente chaque ressource du graphe comme un nœud, OM ne le fait qu'avec les classes. Cela permet de simplifier la vue. Le code de couleurs par type de ressource n'est alors plus nécessaire, mais un code de couleurs basé sur les types de l'ontologie de domaine pourrait s'avérer intéressant et être implémenté de plusieurs manières. Il pourrait, par exemple, se baser sur les ballots ou sur les cardinalités.

Tabulator (Berners-Lee, *et al.*, 2006) prend la forme d'un navigateur RDF générique qui montre les données dans une vue en arbre de laquelle on peut graduellement s'enfoncer dans différents niveaux de détails en cliquant sur les nœuds (voir Figure 6.18). Malgré l'aridité apparente de l'interface, OM permettrait un résultat similaire si les toutes les branches du graphe contenu dans la grille étaient systématiquement présentées et que les cellules redondantes étaient fusionnées.

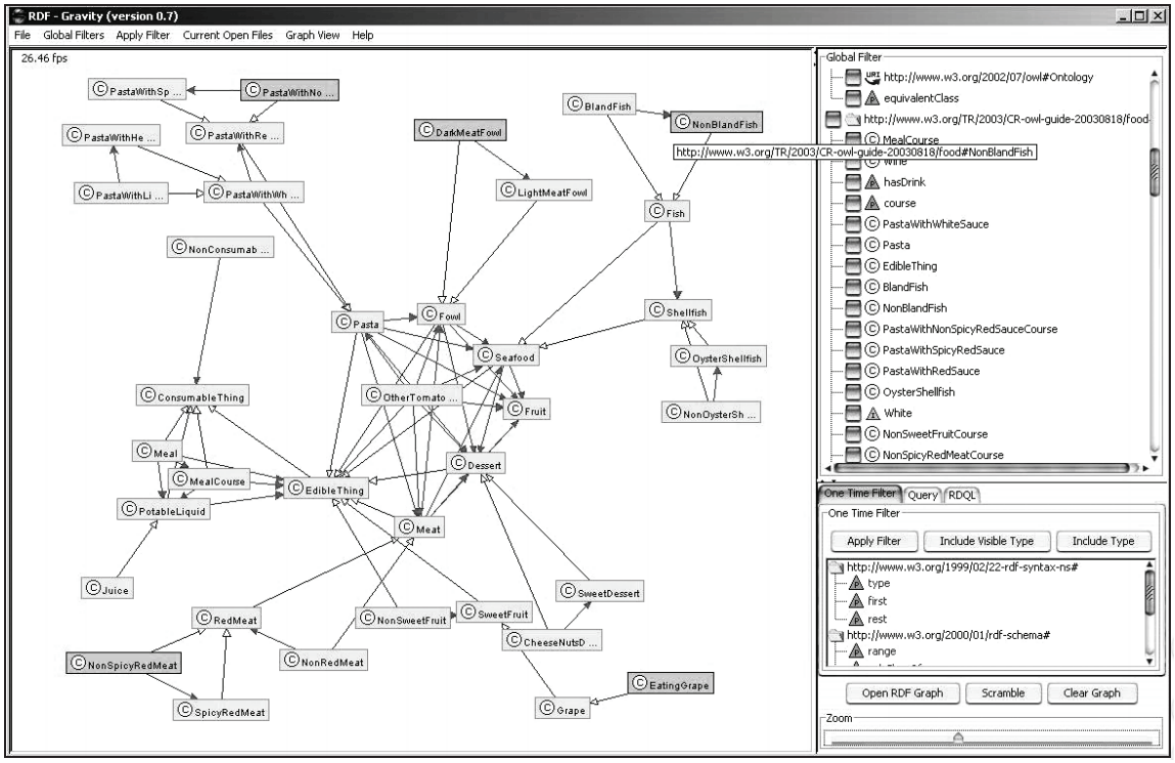

Figure 6.11 RDF Gravity
Source : Tiré de RDF Gravity (2020)

Marie et Gandon présentent aussi une application mobile Web, DBpedia Mobile (Becker & Bizer, 2008), qui se base sur la géolocalisation de l'usager pour présenter des ressources de DBPedia qui se trouvent à proximité sur une carte (Figure 6.12). Des stratégies de filtrages et de compréhension du contexte sont mises en œuvre pour réduire la surcharge d'information. La programmation de composants dirigés par les ontologies utilisant de telles stratégies pour comprendre les données de géolocalisation pourraient être un ajout intéressant à OM.

LENA (LENA, 2020) construit des canevas de visualisation à partir de requêtes SPARQL en se basant sur le langage Fresnel (Fresnel, 2020) (faisant partie des projets SIMILE (SIMILE, 2020)) dans le but de construire des vues répondant aux intérêts et expertises de l'usager (Figure 6.13 LENA). Quoi que LENA offre un style de présentation bien différent d'OM, les deux approches ont des similitudes puisqu'elles font toutes deux usages de canevas de visualisation génériques pour générer la visualisation.

Figure 6.12 DBPedia Mobile

Source : Tiré de Becker & Bizer (2008)

Figure 6.13 LENA

Source : Tiré de LENA (2020)

Dans la catégorie des navigateurs à facettes, Longwell (Veres, Johansen, & Opdahl, 2010), qui fait aussi partie des projets SIMILE (SIMILE, 2020), utilise des heuristiques qui peuvent être préconfigurées pour déduire les facettes et leurs valeurs dans de larges jeux de données (Figure 6.14). Ce genre de mécanisme pourrait être utilisé par OM pour permettre l'exploration de jeu de données dont le schéma est inconnu.

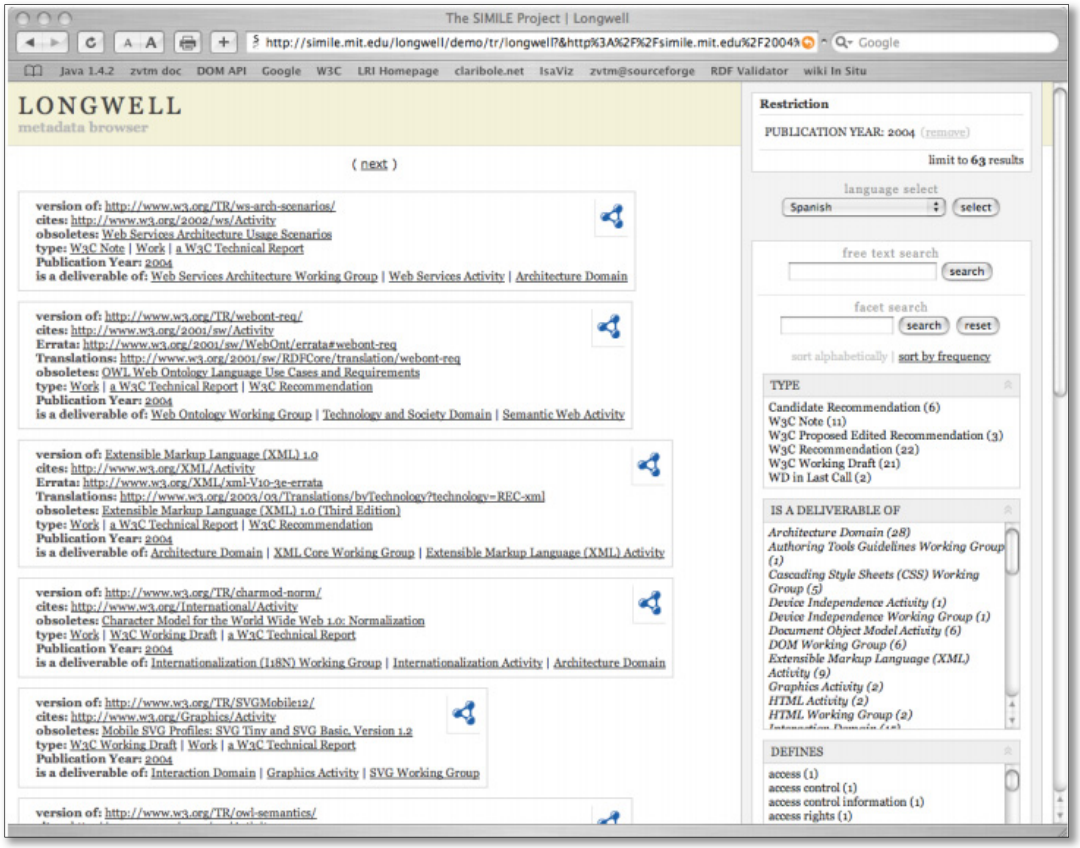

Figure 6.14 Longwell

Source : Tiré de Veres, Johansen, & Opdahl (2010)

mSpace (Wilson, Russell, Schraefel, & Smith, 2006) organise les facettes ordonnées horizontalement par niveaux de profondeur dans la hiérarchie ontologique. Les facettes choisies en premier déterminent les valeurs des facettes choisies par la suite. Un tel mécanisme pourrait être implémenté dans le CRV-HQ par le retrait des propriétés menant à un ensemble vide de la requête. Dans le cas de OM, ce mécanisme pourrait se traduire par la non-possibilité d'ajouter un filtre sur les propriétés non-instanciées.

MuseumFinland (Hyvönen, 2004) présente le nombre de résultats associés à une facette pour aider l'utilisateur à faire des choix (Figure 6.15). Cette fonctionnalité qui était implémentée dans le CRV-HQ manque toujours à OM.

Figure 6.15 MuseumFinland

Source : Tiré de Hyvönen (2004)

Parmi les autres paradigmes de navigation explorés, Parallax (Huynh & Karger, 2009) introduit la navigation par ensemble permettant d'explorer plusieurs liens liant un ensemble à un autre comme par exemple tous les présidents des États-Unis à tous leurs enfants. Une telle fonctionnalité demanderait un grand effort à implémenter puisqu'il remet en question l'utilisation d'une seule classe pivot qui est à la base de l'engin.

RelFinder (RelFinder, 2020) montre le chemin ontologique, au niveau des individus et non du modèle, qui existe entre deux ressources d'intérêts sous forme de graphe (Figure 6.16). De tels mécanismes de navigation par les propriétés et par les individus sont possibles dans OM.

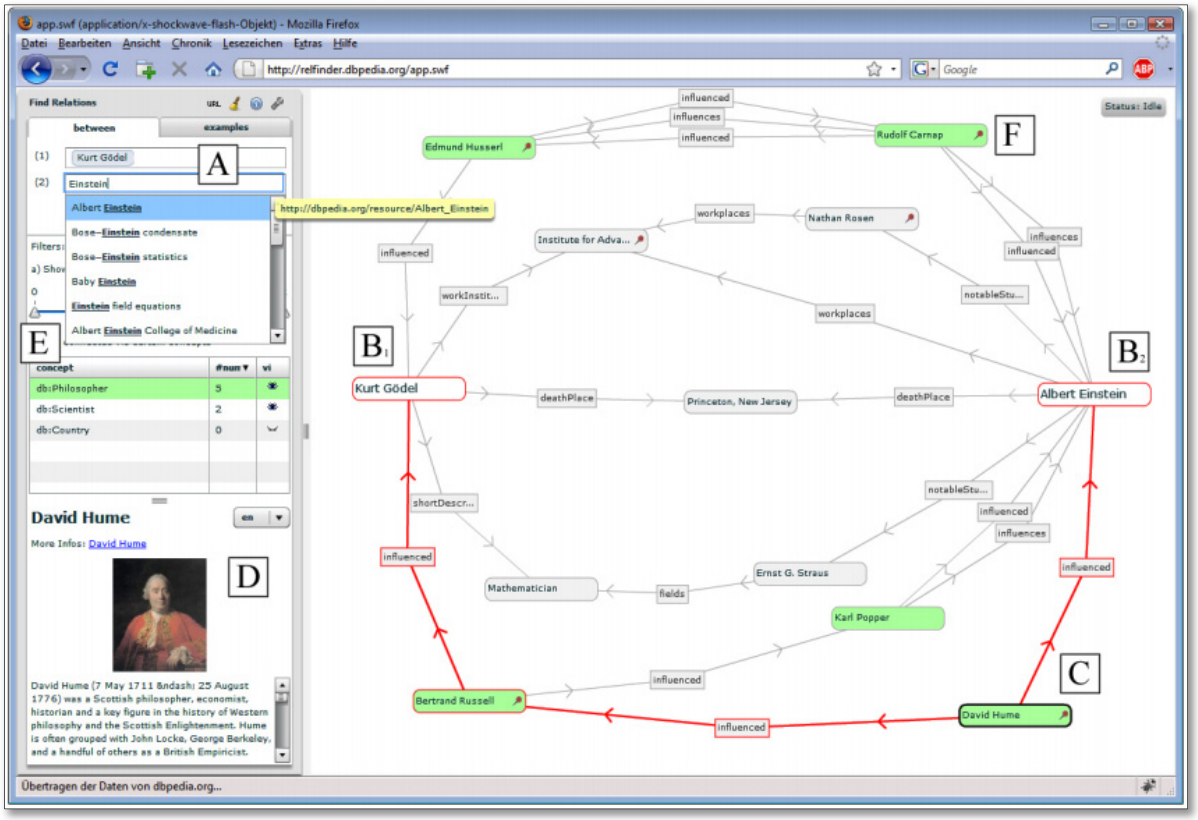

Figure 6.16 RelFinder

Source : Tiré de RelFinder (2020)

Finalement, Visor (Popov, Schraefel, Hall, & Shadbolt, 2011) est un navigateur du WS qui permet une navigation par pivots multiples en suivant plusieurs propriétés à la fois. Il travaille surtout au niveau du modèle, les individus n'étant montrés que sur demande (Figure 6.17). Tout comme Parallax, la remise en question du pivot unique rend difficile cette implémentation dans OM.

Ces différentes fonctionnalités sont regroupées dans le Tableau 6.10.

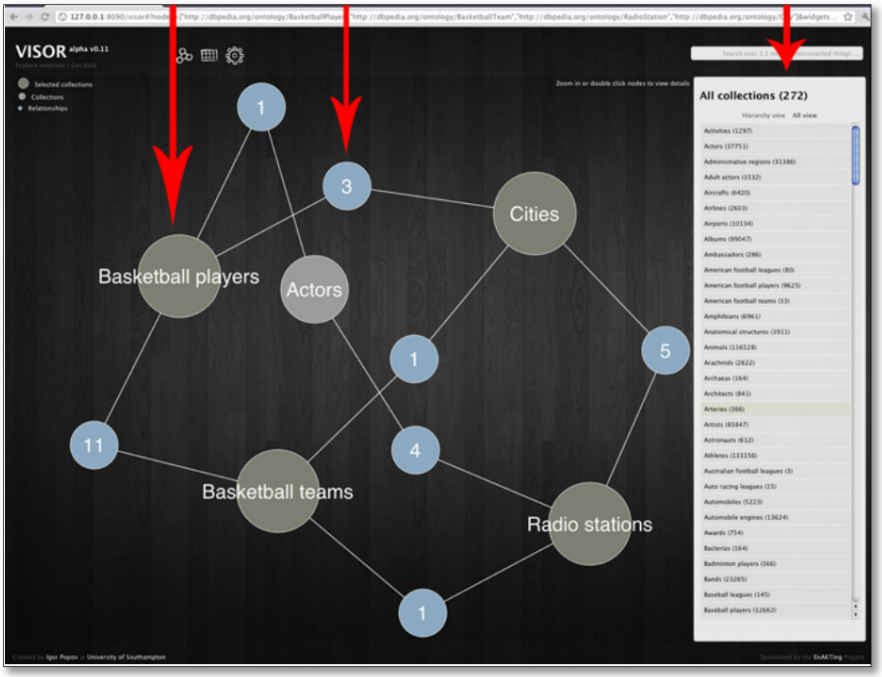

Figure 6.17 Visor

Source : Tiré de Popov, Schraefel, Hall, & Shadbolt (2011)

Tableau 6.10 Présence des fonctionnalités d'autres outils pour données liées dans OM

Outils	**Fonctionnalités présentes dans d'autres outils pour DOL**	**Présence**	**Effort**
RDF Gravity	Utiliser le filtrage, la recherche et le repositionnement des nœuds	Oui	
	Utiliser un code de couleur par type de ressource	Non	Petit à Moyen
Tabulator	Présenter des grilles imbriquées	Non	Petit
DBpedia Mobile	Comprendre les données de géolocalisation	Non	Moyen
	Mettre en place des stratégies de filtrage et de compréhension du contexte	Non	Moyen à grand
LENA	Construire des canevas de visualisation à partir de requête SPARQL	Oui	
Longwell	Utiliser des heuristiques pour déduire les facettes	Non	Moyen à grand
MuseumFinland	Montrer le nombre d'individus par classe	Non	Petit
mSpace	Organiser les facettes ordonnées horizontalement par niveaux de profondeur dans la hiérarchie ontologique	Non	Moyen
Parallax	Permettre la navigation par ensemble	Non	Très grand
RelFinder	Permettre la navigation par les propriétés	Oui	
Visor	Permettre la navigation par pivot multiple	Non	Très grand

6.1.3.3 Exigences, lignes directrices et fonctionnalités de l'étude de Dadzie et Rowe

Tel que mentionné précédemment, plusieurs cadres analytiques sont proposés par Dadzie et Rowe (Dadzie & Rowe, 2011) afin d'analyser les applications pour données liées.

Ces exigences, lignes directrices et fonctionnalités sont reproduites dans les tableaux suivants et discutées en lien avec OM.

En ce qui concerne les exigences que tout système de visualisation devrait combler, le besoin d'une vue générale est établi dans OM comme dans le CRV-HQ par l'utilisation d'un arbre montrant la liste des classes du modèle. Une différence existe cependant dans OM où cette liste peut prendre deux formes. La première unie les classes de l'ontologie et les vues sauvegardées, ces dernières pouvant être considérées comme des classes avec restrictions faisant partie de l'ontologie. La deuxième forme ne comporte que les vues sauvegardées et correspond à la vue générale des utilisateurs normaux qui n'ont pas accès aux fonctionnalités avancées d'OM. Tout comme pour le CRV-HQ, ces vues générales gagneraient à être accompagnées d'outils plus sophistiqués comme une représentation graphique globale de l'ontologie. Ces outils sophistiqués ne sont utiles cependant qu'aux super-utilisateurs, la vue globale des utilisateurs semblant être suffisante pour leurs besoins.

Des filtres peuvent s'appliquer sur n'importe quelle colonne de la grille et la sélection de ceux-ci se fait en ajoutant des classes à la grille et en affichant ou en cachant leurs colonnes.

À la manière du CRV-HQ, des détails sur demande sont disponibles sous la forme de bulles contextuelles informant sur les classes et les propriétés. Cependant, on n'a pas besoin ici d'un service tiers pour fournir des exemples de modalités puisque l'ensemble de solution est constamment visible dans la grille.

On utilisera le même argumentaire que pour le CRV-HQ en affirmant qu'OM permet la manipulation des données multidimensionnelles, hiérarchiques et de réseaux puisqu'elle permet d'afficher les données de n'importe quelle ontologie. La grille sémantique permet en outre plus de contrôle sur leur représentation en deux dimensions.

L'identification et le surlignage des relations entre les données se fait soit par la grille, soit par la composante graphique à travers des outils sur les propriétés de données et objets.

L'extraction des données de la grille se fait en format CSV et en format PNG pour les vues en formulaires. Une extraction au format RDF est toujours absente mais serait très facile à implémenter.

Tel que pour le CRV-HQ, l'historique des vues est disponible à travers les sauvegardes. OM fournit également un historique sur les individus en enregistrant de manière systématique la date, l'heure et le nom de l'auteur lors de leur création. Des mécanismes pour conserver l'historique de chaque modification de chaque individu devrait aussi être conçus pour pousser cette fonctionnalité à sa conclusion logique.

Le Tableau 6.11 résume ces exigences et leur présence dans OM.

Tableau 6.11 Présence des exigences d'un système de visualisation et des lignes directrices de conception dans OM

Fonctionnalités de tout système de visualisation	**Présence**	**Effort**
Une vue générale	Oui	
Des filtres et des possibilités de sélection	Oui	
Des détails sur demande	Oui	
La manipulation de données multidimensionnelles	Oui	
La manipulation de données hiérarchiques (arbres)	Oui	
La manipulation de données de réseaux (graphes)	Oui	
L'identification et le surlignage des relations entre les données	Oui	
L'extraction des données	Limitée	Petit
L'historique	Limitée	Moyen

En ce qui a trait aux exigences des applications sémantiques pour utilisateur débutant, OM permet une navigation intuitive à travers un large graphe de données complexes et multidimensionnelles. Des possibilités de recherche exploratoire sont présentes. Le support pour des requêtes simples est mature, mais celui pour des requêtes avancées est limité. Plusieurs des sémantiques de SPARQL ne sont pas disponibles et de grands efforts devront être fournis pour permettre ces fonctionnalités à travers des composantes qui continueront de protéger les utilisateurs de leur complexité.

Les résultats de la requête présentées dans la grille peuvent être considérés comme l'analyse détaillée des régions d'intérêts; OM remplit donc très bien cette exigence. La publication des données est possible tant par l'ajout de données à la BDT au travers des formulaires CRUD

que par le partage des vues sauvegardées. L'extraction de données est possible, sous format CSV et PNG. La présentation des résultats des analyses peut se faire grâce à la grille mais aussi grâce à l'onglet des graphiques.

Ces exigences et leur implémentation dans OM sont résumées dans le Tableau 6.12.

Tableau 6.12 Présence des exigences des applications sémantiques pour utilisateur débutant dans OM

Fonctionnalités des applications sémantiques pour utilisateur débutant	**Présence**	**Effort**
Navigation intuitive d'un graphe de données complexes et multidimensionnelles	Oui	
Possibilités de recherche exploratoire des connaissances	Oui	
Support pour des requêtes simples et avancées	Limité	Grand
Analyse détaillée des régions d'intérêts	Oui	
Publication des données	Oui	
Extraction des données	Limité	Petit
Présentation des résultats des analyses	Oui	

Les exigences des applications sémantiques pour utilisateur expert commencent par la nécessité d'une navigation intuitive des données liées, ce que permet OM. L'exploration des données permettant de saisir leur structure et leur contenu est aussi présente et *a fortiori* l'exploration des liens à travers le graphe.

Tout comme pour le CRV-HQ, l'exploration pour détecter les erreurs et les anomalies n'a pas du tout été considérée dans le développement d'OM. Le support pour des requêtes avancées utilisant une syntaxe formelle non plus.

Dans cet outil, la publication de nouvelles données est possible et leur validation est faite à plusieurs niveaux. Tout d'abord, l'UI contraint l'usager dans les possibilités de création et de modification de valeurs. Ces contraintes proviennent des types de données et des cardinalités présentes dans l'ontologie de domaine ainsi que des spécifications particulières telles que les maxima et minima ou les exigences d'unicité provenant de l'OV. Le serveur procède ensuite à d'autres vérifications, telles que la préservation de la consistance logique. Idéalement, la validation logique devrait être complètement prise en charge par l'engin d'inférence, mais l'absence de chaînage arrière rend impraticable une telle approche dans OM étant donné le temps de calcul requis.

Encore une fois, l'extraction des données se fait en format CSV ou PNG et le format RDF manque à l'appel. Ces exigences sont résumées dans le Tableau 6.13.

Tableau 6.13 Présence des exigences des applications sémantiques pour utilisateur expert dans OM

Fonctionnalités des applications sémantiques pour utilisateur expert	**Présence**	**Effort**
Navigation intuitive dans les données liées	Oui	
Exploration des données permettant de saisir leur structure et leur contenu	Oui	
Exploration des données pour identifier les liens à travers le graphe	Oui	
Exploration des données pour identifier les erreurs et les anomalies	Non	Moyen à grand
Support pour des requêtes avancées utilisant une syntaxe formelle	Non	Grand
Publication des données	Oui	
Vérification et validation des données	Oui	
Extraction des données pour qu'elles puissent être utilisées dans d'autres systèmes	Limitée	Petit

Finalement, Dadzie et Rowe énumèrent une liste de fonctionnalités que peuvent présentées les applications consommant des données liées.

La première concernant l'utilisation de canevas de présentations représente très bien la technique utilisée dans OM pour présenter les données selon leur type. Ces canevas sont implémentés dans les différentes composantes Web qui s'adapteront aux types des données telles que les colonnes de la grille, les objets Web de filtres et les champs des formulaires. Ces composantes sont aussi dirigées par les sémantiques de l'OV dans une optique d'IDO.

Les points d'entrée dans les données liées sont identiques à ceux du CRV-HQ en ce qui a trait aux super-utilisateurs. Le constat est donc le même : des outils plus avancés pourraient bénéficier aux super-utilisateurs d'OM. Cela dit, pour les utilisateurs normaux, leurs points d'entrée sont les vues sauvegardées et celles-ci répondent très bien à leur besoin.

La généricité est au rendez-vous et les seules entraves à celle-ci sont énumérées dans la section Obligations de modélisation. De plus, la navigation par facettes supporte la recherche intuitive dans les données et la publication de nouvelles données est possible.

Les fusions entre différents jeux de données et entre différentes ontologies ne sont toujours pas implémentées, mais elles sont possibles, envisageables et souhaitables.

L'indication de la provenance des données est implémentée sous le mécanisme d'enregistrement du créateur de chacun des individus du ABox. Cela permet une certaine imputabilité de la qualité de la donnée et donc de la confiance qu'on peut avoir en elle.

L'édition des données est possible. La réutilisabilité des extrants a déjà été discutée précédemment.

Ces fonctionnalités et leur implémentation dans OM sont listées dans le Tableau 6.14.

Tableau 6.14 Présence des fonctionnalités des applications de données liées dans OM

Fonctionnalités des applications de données liées	**Présence**	**Effort**
Des canevas de présentations permettant d'associer les types de données à des standards de présentation	Oui	
Des points d'entrée dans les données liées, par une URI ou par des mots-clés ou encore grâce à une carte	Limitée	Moyen à très grand
Une généricité permettant à l'outil de parcourir des données de différents domaines	Oui	
La recherche et la navigation par facettes pour supporter une recherche intuitive de données multidimensionnelles et mettre en évidence des relations cachées	Oui	
La publication de nouvelles données	Oui	
La fusion de différents jeux de données	Non	Petit
L'indication de la provenance des données et de la confiance qu'on a en elles	Limitée	Moyen
L'édition des données pour pouvoir les enrichir et les corriger	Oui	
La réutilisabilité des extrants	Limitée	Petit

6.2 Enjeux techniques

Les développements de DATE, du CRV-HQ et d'OM se sont heurtés dans chaque cas à des enjeux techniques qui ont dû être étudiés et auxquels des solutions ont dû être trouvées. Ces enjeux et les choix de conception qu'ils ont entraînés seront maintenant discutés.

Le développement de DATE a été ponctué par des premières confrontations avec les enjeux de l'exploration ontologique, de la communication entre le client et le serveur, de la visualisation des données sémantiques et de la construction des requêtes génériques.

Pendant le développement du CRV-HQ, de nouveaux enjeux techniques liés à l'utilisation d'un engin d'inférence, à la sauvegarde et l'optimisation des requêtes et à la pagination ont fait leur apparition.

Finalement, la réalisation d'OM a aussi entraîné toute une nouvelle panoplie d'enjeux techniques qui ont dû être pris en compte dans son développement. Parmi ceux-ci on retrouve les défis techniques liés à la représentation visuelle, le choix d'une vision OO ou mixte lors de l'affichage des données, les possibilités d'ajustements aux changements dans le modèle de données, les possibilités de création des vues et la complexité de l'outil.

6.2.1 Exploration ontologique

La principale limitation de DATE réside dans la manière dont elle explore le modèle de données à chaque requête. Elle ne peut que retrouver les individus des classes ayant une relation associative directe avec un individu sélectionné.

En effet, la construction des arbres n'étant pas générique, c'est l'ontologiste qui implicitement décide des requêtes permises en définissant la forme de ces arbres. Afin d'augmenter la généricité de DATE, il conviendrait de trouver des mécanismes permettant une construction automatisée des arbres. Ceux-ci représentent un facteur crucial pour rendre l'application plus versatile et utilisable avec des ontologies plus complexes.

Pour y arriver, plusieurs avenues sont possibles. Une première serait d'offrir des fonctionnalités dans le UI permettant aux usagers d'explorer l'ontologie et de construire par eux-mêmes les arbres désirés. Cette façon de faire impliquerait de continuer à offrir des arbres présentant des individus de la base de données, tels que les postes, les transformateurs ou les dates de mesures.

Cependant, comme il a été présenté précédemment, les arbres des applications subséquentes ont évolué et ne contiennent plus les individus du modèle, mais bien ses classes. Une deuxième avenue serait donc de ne permettre l'exploration ontologique que par le modèle de données et non plus par ses individus.

Des contraintes sur l'exploration des données comme celles mentionnées à la section Représentations visuelle du CHAPITRE 3, disparaissent ainsi par elles-mêmes. En effet, il n'est dans ce cas plus question de ne chercher uniquement que les classes *sujets* d'une classe *objet*, mais bien de pouvoir parcourir toutes les classes décrivant la classe pivot telles que

décrites dans l'ontologie. Le modèle peut ainsi être parcouru et visualisé en entier selon la volonté de l'utilisateur. Cette façon de faire est implémentée dans le CRV-HQ et OM.

Cela dit, le choix entre effectuer l'exploration ontologique par le modèle ou par ses individus dépend en essence du but de l'application. Un navigateur sémantique tel que DATE accompli très bien la tâche qu'il est censé faire, malgré la non-généricité des arbres.

Lorsque la navigation est générique, les facettes servant à l'exploration peuvent être des individus, des propriétés ou des classes. Des navigateurs à facettes tels que Longwell, et mSpace se servent des individus tandis que MuseumFinland et /facet utilisent les propriétés. Humboldt utilise quant à lui les classes comme facettes.

DATE n'est cependant pas un navigateur par facette. Il se rapproche plus des navigateurs sémantiques textuels où de l'information sur une ressource est obtenue lorsque celle-ci est sélectionnée. Il a la particularité d'être spécialisé pour montrer ces informations sous forme de grille, le rendant particulièrement utile pour représenter des données scientifiques.

Tel que mentionné, l'exploration ontologique se fait différemment dans DATE que dans le CRV-HQ. Alors que la première proposait une exploration par les individus, le second propose une exploration par le modèle.

L'arbre du CRV-HQ sera différent selon que l'usager utilise le mode partiel ou le mode complet. Ces deux modes permettent donc une exploration sensiblement différente de l'ontologie. Le CRV-HQ permet de composer une requête effectuée sur une agrégation virtuelle du graphe du modèle (TBox) et du graphe des individus (ABox) sélectionnés par l'usager. Le mode complet permettra d'explorer l'ensemble de l'ontologie, c'est-à-dire que l'arbre présentera tous les ballots et classes contenus dans l'ontologie et que les boîtes des classes montreront toutes les propriétés de ces classes présentes dans l'ontologie. Le mode partiel, quant à lui, ne montrera que les ballots, classes et propriétés qui sont instanciés dans l'ABox.

Cela implique qu'en mode partiel, l'exploration d'un même TBox se fera différemment lors du changement de l'ABox associé. En revanche, en mode complet, l'exploration du TBox sera la même indépendamment du choix de l'ABox.

Ces deux modes ont leur utilité. Alors que le mode complet permet d'explorer l'ontologie pour mieux la saisir, le mode partiel, lui, permet de plus facilement saisir les données disponibles.

Le mode complet a le désavantage de pouvoir mener à des requêtes dont l'ensemble de solution est vide. Cela dit, d'autres fonctionnalités, telles que la présentation du nombre d'individus contenus dans une classe, permettent d'avoir une idée de la taille de l'ensemble des résultats. En revanche, ces outils n'assurent pas que les requêtes ne retourneront pas un ensemble nul, peu importe que l'usager soit en mode partiel ou en mode complet.

En ce qui a trait à OM, il permet aussi d'explorer les données liées en se servant de leur ontologie pour guider leur exploration, leur présentation et leur utilisation. Cette technique est recommandée par plusieurs auteurs puisqu'elle permet d'aider à l'interprétation du contenu et à l'identification des liens implicites et explicites entre les données (Dadzie & Pietriga, 2017). Dans OM, cela permet de pouvoir présenter des classes ne possédant aucun individu et des propriétés jamais utilisées afin de permettre à l'utilisateur de rajouter de telles informations à la BDT.

Dans OM, l'exploration ontologique se fait par les propriétés objets des classes. Une complexité émerge alors dans la grille sémantique lorsque plusieurs individus sont référencés par une même propriété objet. Ces individus apparaissent tous dans le menu contextuel de la cellule concernée. Afin de pouvoir visualiser tous ces individus simultanément dans la grille, le mécanisme pour copier des lignes a été implémenté.

Pour illustrer la problématique qui en découle, posons l'exemple suivant. La grille présente la classe des Personnes. Il y a une personne par ligne de la grille. Christian a trois amis. Sur l'ajout de la classe des Amis à la grille suivant la classe des Personnes, la ligne présentant Christian pourra être copiée trois fois, chaque ligne présentant un de ses amis différents.

Copier ainsi une ligne entraîne la répétition de toutes ses valeurs avant les colonnes de la classe où les branches divergent. Dans l'exemple précédent, les propriétés de la classe des Personnes sont identiques pour les trois lignes copiées représentant Christian. Tel qu'un SELECT avec une jointure en SQL pourra ramener des lignes dont plusieurs cellules ont la même valeur, ce problème est directement lié à la représentation en deux dimensions de données multidimensionnelles.

Il aurait été possible de remplacer ce dédoublement d'information dans l'interface par des cellules vides ou par la fusion des cellules de mêmes valeurs. Cependant, une telle grille peut rapidement devenir très clairsemée et il a été décidé de ne pas procéder de cette manière.

Un choix a aussi dû être fait sur le comportement par défaut de la grille à la suite de l'ajout d'une classe non-pivot à la grille. La première possibilité aurait été de systématiquement copier toutes les lignes pointant vers plusieurs branches de la nouvelle classe. L'implémentation d'un tel mécanisme par défaut a cependant été rejetée. En effet, celui-ci aurait pu avoir pour effet d'entraîner l'ajout massif de lignes présentant des valeurs redondantes à la grille. Cela aurait aussi pu avoir pour effet de complexifier la compréhension des opérations et augmenter la charge visuelle. De rejeter un tel comportement par défaut permet d'assurer la conservation de la hauteur de la grille à un minimum et par conséquent de conserver aussi la charge visuelle et intellectuelle à un minimum. Dans un cas comme dans l'autre, l'usager peut par lui-même ensuite composer la vue qu'il désire, mais, tel que prescrit par le mantra de Shneiderman (1996), les détails sont disponibles sur demande et non par défaut.

Un corollaire de ce problème est de croiser les données de plusieurs classes dans une seule grille. Une stratégie de grille imbriquée aurait pu être envisagée mais les fonctionnalités et le but d'un tel outil seraient alors différents de ceux poursuivis par OM. Tabulator (Berners-Lee, *et al.*, 2006) est un tel outil. La Figure 6.18 montre un exemple de vue créée avec cet outil.

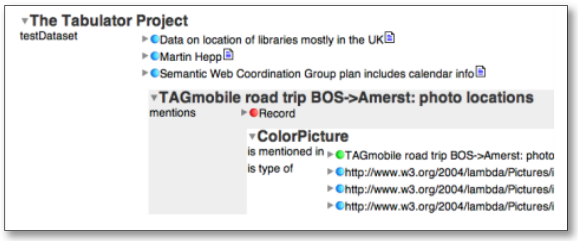

Figure 6.18 Tabulator

Source : Tiré de Berners-Lee *et al.* (2006)

6.2.2 Objet générique

Pendant le développement de DATE, deux véhicules génériques de communication assurant le transfert d'information entre le client et le serveur ont été testés. Le premier était constitué d'un objet Java générique composé par des requêtes SPARQL de type SELECT et transformé au format JSON. Le second utilisait des requêtes SPARQL de type CONSTRUCT pour construire des modèles de triplets envoyés au client sous le format JSON-LD. Pour ces deux approches, les données étaient les mêmes et elles étaient organisées dans la même structure de manière générique, apportant donc la même flexibilité à l'application. À l'exception du système de communication, toutes les autres composantes de l'application sont restées identiques pour ces deux configurations. Les deux systèmes ont été comparés pour leur vitesse et pour la complexité mais aucune de ces métriques n'a apporté d'argument décisif vers l'utilisation d'une approche par rapport à l'autre. C'est lors du développement des applications subséquentes, plus évoluées, que le second système s'est avéré plus avantageux que le premier.

Un des avantages majeurs du second système est que l'objet générique peut évoluer de la même manière que le reste de l'application auto-adaptative. En effet, en utilisant le modèle à base de JSON-LD, il ne suffit que de rajouter des composantes à l'application et de modifier l'OV pour faire évoluer les applications. Cette technique a été abordée au CHAPITRE 5 puisqu'elle a été implémentée dans OM.

Le second avantage s'est présenté tout d'abord dans le CRV-HQ, où le véhicule générique est utilisé pour envoyer la requête visuelle classe par classe à l'engin récursif qui les transforme en une requête SPARQL. Puisque le véhicule générique contient alors tous les éléments nécessaires pour reproduire la requête, il suffit de lui ajouter certaines informations du UI telles que le positionnement des boîtes ou la taille de celles-ci pour qu'il puisse être utilisé pour charger cette requête dans l'interface. Le véhicule devient ainsi le format de sauvegarde des requêtes. Les requêtes sauvegardées par les usagers sont donc essentiellement des copies de ce véhicule générique en format RDF persistées directement dans la BDT. De la même manière, dans OM il servira à l'enregistrement et au chargement des vues alors que lui seront ajouté des triplets contenant l'état exact de la grille sémantique.

Ceci entraîne un avantage collatéral : les vues et les requêtes générées interactivement par les utilisateurs sont enregistrées sous forme de triplets dans un sous-graphe de sauvegardes. Ces enregistrements contenant les URI des classes, propriétés et individus du modèle, il devient aisé de déterminer quels enregistrements utilisent quelles composantes de l'ontologie de domaine. Ils permettent alors d'effectuer la validation automatique des applications utilisant ces vues ou requêtes lorsque des modifications sont apportées à l'ontologie. Ainsi, les changements apportés à l'ontologie qui ne peuvent pas automatiquement être traités par l'auto-adaptivité des applications peuvent tout de même être automatiquement évalués pour établir les bris qu'ils causeront. En plus de pouvoir aisément composer des requêtes afin d'analyser les sauvegardes, on pourra en outre faire des statistiques sur l'utilisation de l'outil et de l'ontologie.

Un autre avantage potentiel du second système est qu'en préservant la structure RDF des données du côté client, on pourra alors harnacher ses capacités en local, telles que l'inférence. Quoi que non testé, cela permettrait de pouvoir utiliser des systèmes à base d'inférence même lorsque la communication à la base de données est coupée, comme dans plusieurs cas d'application lors de la maintenance des réseaux électriques en région éloignée.

Dans le CRV-HQ et OM, nous verrons cet objet s'améliorer de plusieurs façons, mais il contient déjà dans DATE les bases qui permettent l'auto-adaptivité.

Tout d'abord, il offre le transfert classe par classe des données d'une vue. Cette granularité ontologique au niveau de la classe est utilisée dans toutes les applications construites dans le cadre de cette recherche. Il sera discuté plus loin de la possibilité d'étendre ce principe aux autres éléments de méta-modélisation, notamment aux propriétés qui sont des citoyens de premier ordre au même titre que les classes dans les langages RDF, RDFS et OWL.

Ensuite, la séparation entre les informations sur les propriétés et les informations sur les individus permet d'utiliser ces ensembles séparément dans les composantes Web. On pourra notamment construire une grille avec des propriétés issues du modèle, même si aucun individu ne possède encore ces propriétés.

Finalement, la catégorie des accès est déjà présente. Celle-ci atteindra son plein potentiel dans OM en permettant de construire dynamiquement les vues dans le client et non dans la BDT. Dans OM, les informations sur les accès seront assimilées aux informations sur la classe.

6.2.3 Optimisation des requêtes

Un des principaux défis du CRV-HQ concerne l'optimisation des requêtes. En effet, comme les usagers peuvent construire un nombre quasi-infini de requêtes différentes, il est difficile de trouver une façon unique de transformer chaque composante de la requête visuelle en langage SPARQL qui soit optimale pour toutes les situations.

Pour ce qui est d'Oracle, le module *Spatial and Graph* est bâti au-dessus d'un engin SQL. Lorsqu'une requête SPARQL est exécutée, elle est tout d'abord traduite en SQL puis soumise à l'optimisateur de requêtes SQL. Ce dernier n'est toutefois pas très efficace en ce qui a trait à l'optimisation des requêtes SPARQL traduites et plusieurs réglages doivent être effectués manuellement dans les requêtes pour obtenir la meilleure performance possible. Le premier exemple évident de cette limitation est que l'ordre des patrons dans les requêtes influence le temps de réponse d'Oracle. L'utilisation d'indices (*hints*) et la substitution de mots-clés par des patrons et des filtres ou d'autres artifices sont autant de stratégies pour influencer la performance des requêtes.

Ainsi, il semble qu'afin de pouvoir atteindre les performances des applications traditionnelles, des applications comme le CRV-HQ et les autres présentées dans la présente recherche devront attendre que les entreprises de bases de données en graphe offrent des fonctionnalités d'optimisation automatiques des requêtes qui soient elles-mêmes plus performantes.

6.2.4 Pagination

Un défi important du CRV-HQ et d'OM, provient de la pagination. En ce qui à l'application DATE, elle utilisait tout d'abord des mécanismes génériques pour concevoir des requêtes précises. Cette façon de faire a évoluée pendant le développement de l'outil vers l'utilisation

de requêtes génériques. Celles-ci pouvant sélectionner n'importe quelle classe, elles ont simplifié la programmation du côté serveur et accéléré l'obtention des résultats. Puisque dans DATE, chaque classe est présentée dans sa propre grille et que l'exploration se fait par les individus, les mécanismes de pagination pour permettre le chargement paresseux des individus par lot est possible à même la requête générique.

En ce qui a trait au CRV-HQ, le problème survient car le serveur doit envoyer dans sa réponse au client des informations pour permettre la pagination. Ces dernières sont essentielles pour assurer une interface graphique performante puisqu'il est nécessaire d'empêcher le chargement dans la page Web d'un trop grand nombre de données.

Ce processus de pagination diminue cependant grandement la performance des requêtes dans la BDT. D'une part, l'engin de la BDT prend plus de temps à effectuer une requête se servant des mots-clés *LIMIT* et *OFFSET*. D'autre part, pour savoir le nombre de résultats totaux de la requête, cette dernière doit être exécutée deux fois : avec la pagination et sans la pagination. Cette double exécution double le temps de requête par le serveur pour l'obtention de la première page.

Dans le cas d'OM, des mécanismes génériques ont été utilisés conjointement avec des requêtes génériques, afin de pouvoir obtenir les fonctionnalités de pagination désirées. Cette façon de faire est visible dans la Requête 5.5 où l'insertion de multiples patrons de requêtes (dans les espaces réservés <requete> et <requeteIO>) est nécessaire afin de ne ramener que les individus d'une classe étant reliés aux individus de la classe pivot présentés dans la grille. Les mécanismes génériques viennent alors construire la chaîne de patrons grâce au chemin des propriétés (un patron par classe entre la classe pivot et la classe désirée). Cette chaîne de patrons est ensuite introduite dans la requête générique.

L'utilisation de la pagination grâce aux mots-clés *LIMIT* et *OFFSET* pour parvenir à un chargement paresseux au niveau des individus et l'utilisation des chemins de propriétés pour parvenir à un chargement paresseux au niveau des classes mènent à une formulation de cette requête que l'engin d'Oracle est incapable de résoudre dans un temps convenable lorsque le jeu de données dépasse un certain nombre de triplets. Un temps considérable a été investi pour

tenter d'améliorer les temps de performance de cette seule et même requête de l'application : la Requête 5.5.

Contrairement aux pratiques usuelles, cette application a été développée dans un environnement de recherche beaucoup plus puissant que l'environnement de production qui devait héberger sa version finale. Ainsi, lors du transfert de l'environnement de développement consistant en un serveur de triplets d'Oracle d'un demi-million de dollars et ayant environ un téraoctet de RAM vers un plus modeste serveur pour la mise en production d'OM, les temps de réponse de la BDT ont considérablement ralenti. Une réécriture majeure du serveur et de ses requêtes génériques a mené à des formes optimisées de celles-ci. Cependant, aucune formulation ni stratégie n'est venu à bout de donner un temps de réponse acceptable à cette requête lorsque la taille de l'ensemble de données dépasse un seuil relativement petit.

Une demi-douzaine de rencontres ont eu lieu avec trois ingénieurs de données seniors d'Oracle assurant le support sur le module *Spatial and Graph* afin d'aller chercher le maximum de rapidité de la part de la BDT. Ces rencontres n'ont pas apporté les améliorations espérées et il a donc été conclu que l'engin d'Oracle n'était pas capable de résoudre ce type de requêtes simples dans des temps acceptables.

Cette requête a été décortiquée à sa plus simple expression afin d'évaluer lesquels de ses éléments provoquaient les problèmes. Il semblerait que l'utilisation des mots clés *LIMIT* et *OFFSET* en combinaison avec des patrons à trois variables mélangent complètement l'engin d'Oracle.

Pourtant, une analyse a montré que la requête ne présente pas de complexité intrinsèque. Il sera donc recommandé d'attendre qu'Oracle dompte son engin de requêtes ou d'utiliser d'autres technologies de BDT telles que Stardog dont il sera question plus loin afin de se lancer dans une implémentation similaire. Une autre solution consisterait à utiliser un serveur suffisamment puissant pour ramener toutes les données en mémoire, ce qui limite cependant la taille du jeu de données pouvant être utilisé.

6.2.5 Oracle 12c vs Stardog

Ces problèmes de performance d'Oracle ont fait considérer l'utilisation de Stardog, le leader actuel en SGBD de BDT, pour l'implémentation d'OM.

Un avantage non négligeable de Stardog est qu'il implémente toutes les sémantiques d'OWL et permettrait donc les sauvegardes en tant que vues avec restrictions dont il a été question plus tôt. De plus, il fonctionne par chaînage arrière, ce qui, comme il a été discuté, est une avenue prometteuse pour l'implémentation de l'inférence dans une application comme OM.

En outre, Stardog est un SGBD conçu exclusivement pour les triplets. Il ne comporte donc pas d'engin de traduction du SPARQL vers le SQL comme Oracle. Cela laisse présager la disparition d'une partie des problèmes de performance rencontrés avec Oracle.

L'utilisation d'Oracle était un choix imposé par l'entreprise. Cela dit, malgré les quelques problèmes, leur logiciel et leur personnel sont hautement professionnels. Étant donné les aspects matures de leur plateforme et leurs précieux conseils nous recommandons aux entreprises voulant implémenter une approche comme celle d'OM de considérer l'utilisation d'Oracle si les points soulevés sont travaillés parallèlement à des améliorations de leur technologie de SGBD. D'autre part, plusieurs tests seraient à effectuer avec Stardog afin de valider qu'il s'agit bien de la plateforme à préférer pour ce genre d'application.

6.2.6 Défis techniques à la représentation visuelle

Tel que présenté dans la revue de littérature, Liu, Cui, Wu et Liu (2014) décrivent cinq défis techniques reliés à la représentation visuelle. Ces cinq défis sont l'utilisabilité, les possibilités de mise à l'échelle visuelle, l'analyse intégrée de données hétérogènes, la visualisation *in-situ* ainsi que les erreurs et l'incertitude.

Quoi que pour Liu *et al.* le défi de l'usabilité se réfère à la difficulté d'évaluer les outils de visualisation, la liberté sera prise de discuter du défi de l'usabilité différemment. Dans le développement d'OM ce défi s'est traduit par la difficulté à rendre simples les mécanismes d'exploration et d'exploitation des données multidimensionnelles sous forme de graphe. En

effet, malgré la simplification des outils, ceux-ci présentent toujours une courbe d'apprentissage plus abrupte que désirée. Différentes stratégies ont été mises en œuvre pour tenter d'aplanir cette courbe, mais le maintien d'une usabilité adéquate pour des utilisateurs néophytes est un effort constant qui pourra être poussé encore plus loin dans les prochaines itérations.

Le défi de la mise à l'échelle visuelle concerne la complexité qu'il y a à représenter beaucoup de dimensions et beaucoup d'individus dans une même représentation. Ce défi a été abordé par la pagination pour limiter le nombre d'individus d'une part et par des recommandations de modélisation d'autre part. Une discussion sur le sujet se trouve dans la section Complexité d'OM. L'échantillonnage, le filtrage, le clustering ou l'analyse en composantes principales, tels que discutés par Liu *et al.* n'ont toujours pas été utilisés à ce stade-ci du développement d'OM, mais leur étude serait intéressante.

Le défi de l'intégration des données hétérogènes est intéressant à aborder puisque les technologies du WS ont été conçues autour du concept d'interopérabilité. Les standards qui y sont véhiculés, s'ils étaient globalement acceptés et utilisés, permettraient d'éradiquer complètement ce défi d'internet. Cette utopie est le rêve du WS, un Web de données interopérables qui remplacerait le Web de documents que nous avons actuellement.

Ce défi peut aussi être vu sous l'angle de l'intégration des données de plusieurs types. Selon la division historique des techniques de visualisation de l'information en visualisation scientifique et en visualisation de l'information, OM se classe dans ce deuxième sous-champ puisqu'il ne permet pas la visualisation des données spatiales. L'outil permet la visualisation de toutes les catégories typiques d'information de ce sous-champ telles que les données multidimensionnelles, les textes, les graphes et les arbres. Cela dit, des données spatiales pourraient être ramenées par les requêtes génériques et des composants Web pourraient être implémentés pour permettre la visualisation de celles-ci. OM chevaucherait alors les deux sous-champs.

De la même manière, dans la catégorisation de Tory et Möller (2004), la grille sémantique permet de visualiser à la fois des séries de valeurs discrètes et continues d'une part et ayant des

spatialisations choisies ou données d'autre part. Ce sont les différents graphiques et autres composantes de visualisation qui viendront ensuite utiliser les uns ou les autres de ces attributs.

Le défi de la visualisation *in-situ* concerne la visualisation en continue (*streaming)* qui n'a pas été abordée dans le cadre de cette recherche.

Finalement, le défi des erreurs et de l'incertitude qui s'intéresse à l'inconsistance et l'incertitude des données a lui été abordé dans cette recherche. En effet, le mécanisme de l'OV des propriétés d'annotations des individus, jumelé à une programmation judicieuse des composantes Web, permet de savoir en tout temps quel individu a été créé par quel utilisateur et à quel moment. Ce mécanisme instaure une stratégie d'imputabilité de la qualité de la donnée. Il pourrait encore être amélioré en lui ajoutant un mécanisme d'historique des modifications et une granularité au niveau de l'objet de chaque triplet au lieu du niveau de l'individu. Des capacités de calculs et d'intelligence artificielle, jumelées aux capacités déjà existantes d'inférence, pourrait amener la validation des données à un autre niveau.

Tel que mentionné dans la revue de littérature, Dadzie et Rowe (2011) identifient cinq défis pour la visualisation des données liées par des utilisateurs néophytes de ces technologies. Ces défis sont reproduits dans le Tableau 6.15.

Tableau 6.15 Défis de la visualisation pour néophytes

Défis de la visualisation pour néophytes
La grande quantité d'informations à laquelle l'utilisateur doit faire face
La surcharge informationnelle due au nombre de liens d'une ressource vers d'autres ressources
La perte de précision et de complétude de l'information due à la production à large échelle des données liées
Les difficultés pour les utilisateurs ne connaissant pas le fonctionnement de ces technologies
Le besoin d'outils permettant aux utilisateurs de construire des requêtes implicitement ou intuitivement

Certains de ces défis se sont aussi révélés pendant le développement d'OM et des moyens ont dû être pris pour les mitiger. Le premier défi concerne la grande quantité d'information à laquelle doit faire face l'utilisateur. Ce défi aurait pu se poser autant au niveau du nombre de classes dans l'arbre, que du nombre de propriétés dans les classes et du nombre d'individus dans la grille.

Dans OM, un trop grand nombre de classes implique un arbre des classes trop volumineux. Dans PPAL, la première application réalisée grâce à OM, le nombre des classes n'a jamais dépassé un seuil critique de surcharge informationnelle. Cependant, afin de guider l'utilisateur à débuter sa recherche, les classes de l'arbre pourraient être regroupées par ballots tel que cela est fait dans le CRV-HQ. D'autres mécanismes tels que la recherche par mots-clés pourraient en outre être utilisés.

Au niveau du nombre de propriétés par classe, le problème ne s'est pas non plus présenté dans le cadre des applications développées à ce jour. Il se traduirait par un trop grand nombre de colonnes présentes dans la grille sémantique. L'usager peut tout d'abord diminuer la charge informationnelle en cachant les colonnes non désirées. De plus, une modélisation judicieuse peut en partie régler ce problème et la façon de faire sera présentée dans les recommandations. Cela dit, elle ne règle pas tous les problèmes et des recherches seraient nécessaires afin de guider de futurs développements aux prises avec ce défi.

Finalement, le problème du trop grand nombre d'individus présentés dans la grille a été abordé par un mélange de pagination et de chargement paresseux. En effet, seulement une trentaine d'individus sont ramenés à la fois de la BDT, par classe. Si l'utilisateur parcours la grille grâce à la barre de défilement verticale, les autres individus sont chargés page par page. La grille se chargeant aussi classe par classe, un chargement paresseux est donc également effectué de manière horizontale. L'usager pourra utiliser les fonctions de tri et de filtres afin de diminuer le nombre d'individus présentés dans la grille.

Le deuxième défi concerne le nombre de liens qui peuvent unir un même individu à plusieurs autres ressources. Il se traduit dans OM par un grand nombre d'individus pouvant apparaître dans le menu contextuel d'une cellule objet. Tel que mentionné précédemment, ce défi cherche toujours une solution élégante. Dans PPAL, le problème ne s'est pas présenté. Une barre de défilement à tout de même été ajoutée aux menus contextuels contenant plus d'une dizaine de liens. Des idées telles que de présenter les liens possibles par ordre décroissant d'utilisation des individus dans les vues sauvegardées et l'ajout d'une interface additionnelle dont les différentes fonctionnalités faciliteraient la sélection pourraient être jumelées afin de répondre à cette problématique.

Le défi de la perte de complétude et de précision due à la production massive de données liées ne s'est pas présenté et aucune piste de solution n'a donc été développée.

Finalement, le but d'OM était de relever les deux derniers défis. Le défi posé par la difficulté des utilisateurs à comprendre les technologies du WS a été adressé en protégeant les utilisateurs de cette complexité par l'utilisation de paradigmes de visualisation et d'interaction simples, connus et intuitifs. Le défi de permettre la composition intuitive de requête a été relevé par l'utilisation simultanée de la grille sémantique et de l'outil de requête graphique.

D'autres composantes ont aussi été dédoublées afin d'augmenter l'utilisabilité d'OM, telles que les formulaires CRUD. Les deux types de formulaires utilisés pour effectuer les modifications dans la BDT, le formulaire à onglets et le formulaire simplifié, semblent tous les deux pouvoir être utiles selon le contexte. Le formulaire CRUD à onglets est plus approprié lorsqu'on est en présence d'une vue composée d'un grand nombre de classes présentant chacune un grand nombre de propriétés, alors que le formulaire CRUD simplifié apparaît être plus approprié lorsque le nombre de classes et le nombre de propriétés sont moins élevés. Dans les situations de terrain, lorsque des usagers doivent effectuer des actions CRUD alors qu'ils effectuent leurs tâches opérationnelles, le formulaire CRUD simplifié s'avère plus simple à utiliser.

6.2.7 Vision OO ou vision mixte

D'une part le cadre de développement Web (Ext.js) et le langage du serveur (Java) sont des langages OO pour lesquels les relations entre les superclasses et les sous-classes passent par l'héritage. Dans le paradigme OO une sous-classe hérite des propriétés de ses superclasses. Les usagers néophytes des technologies sémantiques sont en général habitués à ce paradigme et s'attendent à ce que les applications présentant des hiérarchies adhèrent à celui-ci dans le comportement des fonctionnalités.

D'une autre part, les technologies sémantiques résident dans un paradigme d'inférence logique suivant les axiomes de la logique de premier ordre. Ce paradigme adopte une vision de classification des individus dans l'espace ontologique, ce qui modifie le sens des relations entre

les classes et les sous-classes. En effet, un individu faisant partie d'une classe fait également partie de ses super-classes et ce sont les super-classes qui héritent des propriétés des sous-classes. Cette approche permet de classifier correctement un individu à partir de ses propriétés connues.

Voici un exemple simple de cette distinction. Dans un paradigme OO, je peux déduire qu'un canard et qu'un éléphant se nourrissent tous deux de substances organiques puisqu'ils sont des animaux. Dans un paradigme de classification logique, je peux déduire qu'un individu tiré au hasard sera un animal s'il se nourrit de substances organiques et il pourra donc posséder un bec ou une trompe et toutes les autres propriétés qu'ont tous les animaux, tant que je ne pourrai le classifier plus précisément.

Tel que discuté à la section Paradigmes de classification et d'héritage, des choix de conception ont dû être faits pour réconcilier les paradigmes OO et de classification dans OM. La Requête 5.4 montre que lorsqu'on ramène les propriétés d'une classe, OM adopte une vision mixte des paradigmes d'héritage OO et de classification logique. En effet, tel qu'expliqué précédemment, la sémantique *rdfs:domain* d'une propriété, après inférence, pointera vers la classe domaine de cette propriété avant inférence et toutes ses sous-classes. C'est pourquoi le jeu de pré-inférences présenté précédemment introduit la sémantique *om:hasDirectDomain* qui relie une propriété à la classe domaine de l'ontologie avant inférence et toutes ses super-classes.

La Requête 5.4 pourtant ne se contente pas de ramener l'un ou l'autre de ces ensembles, mais ramène les deux. C'est ce que nous nommons « adopter une vision mixte de ces paradigmes ».

De premier abord, l'implémentation pourrait se passer des propriétés qui ont comme domaine la classe demandée (classification) et se contenter de celles qui l'ont comme domaine direct (OO). N'utiliser que le domaine direct implique qu'uniquement les propriétés associées à une classe qui sont communes à toutes les sous-classes sont présentées dans la grille sémantique, même si des individus proviennent de ces sous-classes.

L'autre choix ramène, comme dans le paradigme OO, les propriétés de la classe, de ses superclasses et, comme dans le paradigme de classification, celles de ses sous-classes. La décision d'aller vers une option plutôt que l'autre devrait se prendre en fonction du modèle de

données. Par exemple, une hiérarchie profonde avec plusieurs niveaux de sous-classes et plusieurs propriétés sur chaque niveau peut bénéficier d'un affichage épuré.

D'autres facteurs sont aussi à considérer dans le choix entre l'adoption d'une vision purement OO et d'une vision mixte. Dans un premier cas, l'utilisateur pourrait ne pas voir toutes les propriétés des individus. Cette approche facilite la compréhension de l'information contenue dans la grille mais pourrait ne pas être souhaitable dans certaines situations. Dans un deuxième cas, toutes les informations du modèle sont présentées. Quoique cela puisse provoquer une surcharge informationnelle, l'utilisateur se trouve alors en possession de toutes les informations disponibles pour comprendre les individus.

Un autre point à prendre en considération est la contrainte sur l'utilisation de l'outil impliquée par le choix de la vision. Dans le premier cas, l'interface contraint l'utilisateur à créer des individus de la classe sélectionnée, mais l'empêche de créer des individus de ses sous-classes sans modifier la vue. Dans le deuxième cas, l'utilisateur peut créer des individus appartenant à une sous-classe de la classe sélectionnée mais aussi des individus appartenant à plusieurs sous-classes simultanément, ce qui peut ne pas être souhaité. En revanche, on peut mitiger ce problème par l'utilisation des sémantiques pour déclarer des classes disjointes dans l'ontologie.

Dans un autre ordre d'idée, une modélisation judicieuse peut classer les propriétés des sous-classes qui ne sont utiles que sur un seul niveau ontologique dans des classes liées par une propriété objet fonctionnelle et inversement fonctionnelle. Cette façon de faire diminuera la charge d'affichage de toutes les classes de la hiérarchie.

Dans OM, la vision mixte a été sélectionnée pour permettre aux individus des sous-classes d'être créés à n'importe quel niveau. Ceci ne permet cependant pas d'empêcher intrinsèquement la création d'un individu faisant partie de deux sous-classes distinctes, comme un affichage purement OO le ferait. Pour ce faire des mécanismes de vérifications ont été ajoutés dans le système.

6.2.8 Solution non retenue de réconciliation OO et classification

Tel que mentionné précédemment, plusieurs techniques ont été explorées pour réconcilier les paradigmes OO et de classification. Dans le CRV-HQ, cette réconciliation a été effectuée grâce à une fonction offerte par le module *Spatial and Graph* d'Oracle. L'utilisation de cette fonction complexifiant inutilement le développement de OM, la technique retenue dans ce cas a consisté au développement d'un jeu de pré-inférences.

Une autre technique a cependant été développée puis rejetée en raison de sa lourdeur computationnelle et donc de la perte de performance qu'elle entraînait. Elle est toutefois digne de mention puisqu'elle présente ses propres avantages. Elle consiste à déterminer les propriétés qui ont comme domaine la classe observée et toutes ses superclasses par une requête SPARQL à chaque demande de l'usager. On détermine au même moment quelle classe parmi celles qui sont codomaines des propriétés est la plus spécialisée dans l'ontologie. La Requête 6.1 montre comment procéder.

```
1  SELECT ?prop ?r ?r2 ?r3 WHERE {
2          <CLASSE> rdfs:subClassOf+ ?dom .
3          FILTER (?dom NOT IN (owl:Thing, rdfs:Resource))
4          OPTIONAL {
5                  ?notDom rdfs:subClassOf ?dom .
6                  FILTER (?notDom NOT IN (<CLASSE>,  ?dom))
7          }
8          BIND( IF (BOUND(?notDom) , ?notDom, owl:Nothing) as ?notDomBinded)
9          ?prop rdfs:domain ?dom .
10         MINUS { ?prop rdfs:domain ?notDomBinded . }
11         {
12                 ?prop a owl:DatatypeProperty .
13                 ?prop rdfs:range ?range .
14                 FILTER (?range NOT IN (rdfs:Literal,owl:Thing, owl:Nothing, rdfs:Resource))
15         }
16         UNION
17         {
18                 ?prop a owl#ObjectProperty .
19                 ?prop rdfs:range ?range .
20                 FILTER (?range NOT IN (owl:Thing, owl:Nothing, rdfs:Resource))
21                 OPTIONAL {
22                         ?prop rdfs:range ?range2 .
23                         FILTER (?range2 NOT IN (owl:Thing, owl:Nothing, rdfs:Resource))
24                         FILTER (?range2 != ?range )
25                 }
26                 BIND( IF (BOUND(?range2) , ?range2, owl:Thing) as ?range3)
27                 FILTER NOT EXISTS { ?range3 rdfs:subClassOf+ ?range .  }
28         }
29 }
```

Requête 6.1 Trouver les domaines et codomaines directs

Cette requête comprend deux actions principales : retrouver les propriétés (?prop) de la classe **CLASSE** grâce à leur domaine (lignes 2 à 9) et retrouver les codomaines de ces propriétés. Il faut garder à l'esprit que cette requête s'applique sur un jeu de données déjà inféré selon les règles de RDFS et d'OWL 2.

Pour retrouver les propriétés, on commence par déterminer la variable ?dom (ligne 2), qui contient toutes les classes qui sont la classe **CLASSE** elle-même (puisque par la règle rdfs6, une classe est sous-classe d'elle-même) et toutes ses super-classes, sauf les classes de haut niveau que sont *owl:Thing*, *owl:Nothing* et *rdfs:Resource*.

Ensuite, on met dans la variable ?notDom toutes les classes qui sont une sous-classe directe de la classe **CLASSE** mais qui ne sont pas la classe **CLASSE** elle-même, ni une classe contenue dans ?dom, ni une des classes génériques *owl:Thing* ou *rdfs:Resource*. La variable ?notDom contient alors uniquement les sous-classes de la classe **CLASSE**. Afin de s'assurer du bon fonctionnement du MINUS (ligne 10), on doit s'assurer que les variables qu'il compare soient toutes les deux populées. Ainsi, la ligne 13 lie (*bind*) une valeur à la variable ?notDomBinded dans le cas où elle n'en a aucune, et les valeurs de ?notDom dans le cas contraire.

Finalement, les propriétés ?prop sont celles dont les domaines sont les classes contenues dans la variable ?dom mais pas dans la variable ?notDomBinded. Nous obtenons donc la liste des propriétés d'une classe au sens du paradigme de l'héritage orienté-objet.

Pour trouver les codomaines des propriétés, nous allons devoir traiter les propriétés de type données (lignes 12 à 14) différemment des propriétés de type objets (lignes 18 à 27).

Pour les propriétés de type données, les codomaines sont simplement les codomaines de la propriété auxquels on enlève les classes génériques *owl:Thing*, *owl:Nothing*, *rdfs:Literal* et *rdfs:Resource*, puisqu'elles sont les seules superclasses du codomaine qui peuvent être déduites par inférence.

Dans le cas des propriétés de type objets, plusieurs codomaines ont pu être inférés. Il faut retrouver celui qui correspond à la classe de l'ontologie la plus éloignée d'*owl:Thing* parmi les classes qui sont listées comme codomaines, puisque par scm-rng1, le codomaine d'une propriété est aussi le codomaine de ses superclasses. Cela revient à trouver le codomaine de la liste des codomaines qui n'a pas de sous-classe. On y arrive en assignant les codomaines d'une

propriété qui ne sont pas *owl:Thing*, *owl:Nothing* ou *rdfs:Resource*, dans deux variables distinctes (?range et ?range2) dont l'une est optionnelle et ne doit pas contenir les valeurs de l'autre. Les ensembles de solution de ?range et ?range2 à ce point sont toutes les combinaisons de codomaines de ?prop où ?range et ?range2 ne sont pas identiques.

On doit s'assurer que la variable optionnelle (?range2) ne contient pas un ensemble vide pour que le filtre de la ligne 27 fonctionne, ce qu'on fait par le *BIND* de la ligne 26, qui assigne *owl:Thing* à ?range3 si ?range2 n'a pas de valeur et les valeurs de ?range2 autrement. Finalement, on retire de l'ensemble des résultats les combinaisons où ?range3 est une sous-classe de ?range. On obtient ainsi uniquement le *rdfs:range* original de la propriété objet.

L'avantage de cette manière de procéder est qu'elle permet d'interroger n'importe quel jeu de données, inférés ou non, sans avoir à faire de transformations. On pourrait ainsi utiliser la méthodologie sur des jeux de données se trouvant à l'extérieur du contrôle de l'entreprise tels que des points de terminaison publics.

6.2.9 Auto-adaptabilité

Dans DATE, l'auto-adaptabilité se fait grâce à des requêtes génériques qui interrogent le modèle et les données simultanément. Elle permet que des modifications soient apportées au modèle de données sans que l'application ne nécessite de modification ou de recompilation. Les modifications permises sont alors restreintes aux classes qui sont directement liées à la classe des feuilles de l'arbre par un lien d'association ainsi qu'aux propriétés de ces classes.

Ce sont aussi des requêtes génériques qui permettent l'auto-adaptabilité d'OM, mais celles-ci ne sont plus contraintes comme dans le cas de DATE. Il convient maintenant de faire un tour d'horizon des changements pouvant être effectués à l'ontologie et d'analyser comment ces changements sont abordés par OM.

Étant donné la généricité des requêtes d'OM, il est possible de charger des vues qui ont été créées sur un modèle de données qui a évolué depuis la sauvegarde. Les changements apportés à l'ontologie se répercutent immédiatement dans les vues sans entraîner la nécessité de

modifier les requêtes de chargement des vues ou les composants Web. Cependant, certains changements peuvent entraîner des modifications aux vues sauvegardées.

Les changements à l'ontologie qui concernent l'ajout de classes ou de propriétés n'ont aucune incidence sur les applications. Les vues sauvegardées vont continuer à fonctionner et les colonnes présentes dans la grille ne changeront pas, puisqu'au chargement d'une vue les nouvelles colonnes sont invisibles par défaut. Les données des nouvelles propriétés seront tout de même ramenées dans le client et l'usager pourra désormais les ajouter à sa vue. L'usager ayant les droits peut alors croiser les données d'une vue existante avec les données d'une nouvelle classe ou d'une nouvelle propriété. L'exploration ontologique permet de voir ces classes et ces propriétés et de les utiliser pour formuler des vues. Les changements aux cardinalités des propriétés, tant qu'ils n'invalident pas logiquement la BDT, se répercuteront aussi dans les vues, sans aucune incidence.

Les changements ontologiques qui consistent au retrait d'une classe ou d'une propriété de l'ontologie n'affectent pas non plus les applications. Cependant, les vues sauvegardées ne présenteront évidemment plus les informations de celles-ci. Tous les composants Web qui se basaient sur ces classes ou sur ces propriétés s'adapteront ou ne seront simplement pas ajoutés à l'UI.

Puisque les sauvegardes des vues contiennent en elles tous les URI des classes et propriétés impliquées, il serait simple d'instaurer des mécanismes pour notifier les propriétaires et les utilisateurs de ces vues lorsqu'un tel changement survient ou est survenu. De plus des mécanismes encore plus évolués pourraient implémenter un système où les parties prenantes des vues impliquées doivent donner leur accord avant qu'un tel changement ne devienne effectif. Ce genre de stratégie pourrait permette d'automatiser un processus communautaire d'utilisateur-décideur où uniquement le temps des personnes concernées est utilisé lorsque de telles décisions doivent être prises. Cela contribuerait à déplacer le pouvoir décisionnel le plus bas possible dans la hiérarchie tel que préconisé par les nouvelles approches de gestion et par l'Industrie 4.0.

Lors de changements aux noms, aux commentaires et aux autres annotations, les applications continuent de fonctionner normalement. En ce qui concerne les noms, cette adaptabilité est due

au mécanisme de division entre les étiquettes lisibles par les machines d'avec les étiquettes lisibles par les humains. Ces premières, les URI, ne changeront pas lorsque les deuxièmes changent, ne provoquant ainsi aucun problème aux applications.

Les changements plus complexes à la structure de l'ontologie provoqueront les mêmes réactions qu'une série d'ajouts ou de retraits de classes ou de propriétés. Ainsi l'ensemble fonctionnera encore mais il est à prévoir que des vues sauvegardées changent. Outre les mécanismes semi-automatisés pour effectuer une gestion de ce changement, des mécanismes plus complexes pour comprendre ces changements et adapter les vues automatiquement doivent faire l'objet d'études plus approfondies pour pouvoir affirmer leur faisabilité et confirmer leur pertinence. Les interactions de complexités différentes entre diverses sémantiques de RDFS ou OWL devraient également être étudiées en profondeur quand des composants Web plus évolués se servant d'elles seront disponibles.

En ce qui a trait aux individus, les vues sauvegardées présentes les individus de la version la plus récente de la BDT. Ainsi, les individus seront filtrés ou non selon ce qui se trouve dans la BDT au moment du chargement de la vue.

Ainsi, les applications générées grâce au cadre s'adaptent par elles-mêmes à plusieurs situations. Pour tous les changements qui ne pourront pas être réalisés automatiquement, des mécanismes semi-automatiques pourront être tout de même développés pour aider à la gestion du changement, proposer des pistes de solution ou mettre à jour les sauvegardes affectées lorsque cela est possible.

L'auto-adaptivité est abordé d'un tout autre point de vue dans le CRV-HQ que dans DATE et OM. Dans ces dernières, ce sont des composantes spécialisées qui permettent des fonctions précises grâce à des requêtes génériques. *A contrario*, dans le CRV-HQ, l'utilisateur peut construire une requête précise grâce à des composantes génériques dont la conception calque le langage de requête.

6.2.10 Complexité d'OM

Deux fonctionnalités peuvent être considérées comme les goulots computationnels d'OM. Ce sont la boucle de génération de la grille du côté client et la requête pour fabriquer l'objet générique du côté serveur (Requête 5.8 à Requête 5.11).

En ce qui concerne la boucle permettant de construire la requête, celle-ci comporte trois niveaux. Elle itère tout d'abord sur chaque objet générique, de classe en classe. Dans chacun de ceux-ci, elle parcourt la liste des propriétés qui correspondent aux colonnes de la grille puis la liste des individus qui représentent les lignes de la grille. Ainsi, chaque cellule de la grille pourra être évaluée comme il se doit : avec une valeur pour les propriétés de type données et avec une liste de paires (URI, étiquettes) pour les propriétés objets.

Les individus de la classe pivot sont tous présentés par défaut en ordre alphabétique tandis que les individus des autres classes dépendent des valeurs présentes dans les cellules des propriétés objets de la classe qui les précèdent faisant le pont entre les deux classes.

Ainsi, la complexité de cette boucle se calcule en additionnant le temps pour traiter chaque classe et ce dernier dépend du nombre d'individus et du nombre de propriétés de celles-ci.

En ce qui a trait à la Requête 5.8, sa complexité dépend aussi du nombre d'individus et du nombre de propriétés. Puisqu'on effectuera une telle requête pour chaque objet générique, on peut affirmer que ces deux goulots ont la même complexité.

Le nombre d'individus à présenter dans la grille est contrôlé à l'aide d'un mécanisme de pagination. Celui-ci permet un chargement paresseux qui limite le nombre d'individus retournés à un nombre présélectionné. Il est effectué grâce aux mots-clés *LIMIT* et *OFFSET* appliqués sur les requêtes SPARQL qui retrouvent les individus.

Le nombre de propriétés d'une classe, quant à lui, peut être limité grâce à une modélisation judicieuse du domaine. Il est en effet recommandé de regrouper les trop nombreuses propriétés d'une classe dans plusieurs classes reliées par une propriété objet fonctionnelle et inversement fonctionnelle lorsque possible. Ainsi, le temps de chargement des classes varie moins et ceci à l'avantage de laisser à l'usager le choix d'afficher chacun de ces sous-ensembles de propriétés dans sa vue, allégeant la charge informationnelle. Cette pratique a donc une influence positive

autant sur l'utilisabilité que sur la mise à l'échelle sans toutefois modifier le sens du modèle. Cela dit, aucune ontologie utilisée dans le cadre de cette recherche n'a présenté de cas où le nombre de propriétés d'une classe dépassait la limite du nombre de triplets pouvant être ramenés en mémoire de manière performante.

6.3 Réflexions théoriques

Des trois applications ont découlé plusieurs réflexions théoriques sur les différents aspects de leur conception, leur développement et leur l'utilisation. Seront maintenant présentées les plus importantes de celles-ci, portant sur l'IDM, l'acceptabilité, l'expressivité et l'utilisabilité, l'IDO, les paradigmes de visualisation et l'Industrie 4.0.

6.3.1 Ingénierie dirigée par les modèles

Pendant la réalisation de DATE, force a été de constater qu'il existe un parallèle entre les notions de l'IDM et l'auto-adaptabilité des applications. Voici un résumé de cette réflexion, adapté de Bhérer, Vouligny, Gaha, & Desrosiers (2016).

Les systèmes d'information utilisant le principe de l'auto-adaptabilité tombent dans le paradigme de l'ingénierie dirigée par les modèles, dans le sens qu'un langage de méta-modélisation est utilisé pour décrire tous ces systèmes d'information indépendamment de leur domaine. Une conception utilisant uniquement des composantes logicielles s'adaptant à l'ontologie suffit donc à générer l'application.

Dans la terminologie de MDA, le système d'information est alors considéré M0; les composantes logicielles dirigées par les ontologies sont considérées M1; le langage OWL est considéré M2 et les langages RDF et RDFS sont considérés M3.

L'implémentation de DATE utilise donc RDF, RDFS et OWL comme modèles indépendant de la plateforme (PIM) et JavaScript comme le langage spécifique à la plateforme (PSL). En encapsulant le PSL dans le PIM, i.e., par l'utilisation de composantes applicatives entièrement gouvernées par l'ontologie, la transformation du modèle devient alors générique. Ce modèle

de transformation fonctionnera donc indépendamment du PSL. En effet, tant et aussi longtemps que l'engin qui parcours l'ontologie et transforme ses informations pour les applications Web utilise des fonctions génériques, il peut les construire dans n'importe quel langage de programmation Web.

6.3.2 Acceptabilité du CRV-HQ

Une des principales contraintes du CRV-HQ était l'acceptabilité de l'outil par les ingénieurs habitués au monde relationnel et aux outils grand public tels que Microsoft Access ou Toad for Oracle. La majorité des articles consultés sur les CRV SPARQL tentent plus ou moins de répertorier les conditions gagnantes pour atteindre l'acceptation de leurs interfaces. Il compare ces conditions en termes de charge mentale, d'utilisabilité, d'utilisateurs cibles, de préférences entre les techniques de visualisation, etc.

Quoique ces recherches soient importantes pour tenter de trouver de nouveaux paradigmes de visualisation, elles ne prennent que rarement en compte le fait que les futurs utilisateurs des technologies sémantiques sont pour la plupart déjà des utilisateurs des technologies relationnelles et, qu'en tant que tels, ils ont déjà élu des paradigmes de préférence. Des outils comme Microsoft Access ou Toad for Oracle ont passé le test du temps, et l'hypothèse qu'ils réunissent en eux les conditions nécessaires pour être adoptés par les communautés semble pouvoir être posée avec confiance. En minimisant le changement qu'a à subir un utilisateur, on minimise aussi sa résistance, encourageant ainsi le passage d'une technologie à l'autre à se faire de manière transparente, conformément au principe de moindre surprise (PoLA, 2020).

L'effort de cette étape de la recherche a donc été mis sur le fait d'offrir les mêmes fonctionnalités disponibles dans ces outils relationnels, sous une forme semblable, dans un constructeur de requêtes sémantiques. Les nouvelles fonctionnalités rendues possibles par le changement technologique y sont aussi considérées puisqu'elles représentent des avantages potentiels sur les technologies relationnelles, particulièrement si elles sont intuitives, utiles et faciles d'utilisation.

6.3.3 Expressivité et utilisabilité

Le véhicule générique apparu dans DATE est aussi utilisé dans le CRV-HQ quoi que dans une moindre mesure. Il contient alors les informations sur les classes et leurs propriétés et sert à instancier les boîtes graphiques. Dans le CRV-HQ, on pourrait se passer de ce genre de véhicule puisque la généricité de l'application provient en grande partie du fait que le système de requête visuelle est très près du langage de requête SPARQL. Ce langage permettant évidemment d'interroger et de naviguer n'importe quel modèle ou jeu de données construit en RDF, l'application hérite *ipso facto* des mêmes capacités.

En un sens, un CRV qui serait collé au langage SPARQL serait d'une généricité parfaite. Son expressivité serait totale, mais son utilisabilité ne serait que très peu supérieure à celle du langage de requête. Par contraste, dans le cas d'une application comme DATE, c'est l'utilisabilité qui est presque maximale puisqu'elle est constituée que de très peu de fonctionnalités et que celles-ci sont très simples. Son expressivité est cependant très limitée. Ayant visité ces extrêmes, OM montre comment un judicieux compromis peut apporter une synergie en prenant le meilleur des deux mondes.

On en revient à la dichotomie SRV et LRV, le LRV calquant le langage et le SRV en sacrifiant l'expressivité pour en augmenter l'utilisabilité. Cette dichotomie peut être représentée sur un spectre, le CRV-HQ se trouvant près des LRV, DATE se trouvant près des SRV et OM se trouvant quelque part entre les deux.

Il serait tentant de classifier le CRV-HQ comme un LRV étant donné sa proximité avec SPARQL. Cela dit, certaines hypothèses de base du CRV-HQ sacrifient de l'expressivité du langage de requête au profit de l'utilisabilité d'une part et de la simplicité d'implémentation de l'autre. Ces hypothèses comprennent la forme en arbre des requêtes par l'utilisation de la classe pivot, l'absence de sous-requêtes, l'absence d'implémentation de certains mots-clés tels que SERVICE, etc.

De plus, les LRV représentent généralement chaque nœud d'un graphe RDF comme une entité propre dans la visualisation. Le CRV-HQ fournit une agrégation simple mais cruciale pour la représentation : les propriétés sont représentées comme des éléments contenus dans les classes

qui sont les nœuds de l'outil d'exploration. De plus, les valeurs associées à ces propriétés ne sont pas visibles dans l'espace graphique. On voit fréquemment des LRV utiliser la nature en graphe du langage RDF comme base de la visualisation. Ils cartographient alors chaque triplet directement dans la représentation visuelle. Ce genre de pratique est remis en question dans la littérature (Karger, 2006), entre autres à cause de la surcharge informationnelle qu'elle entraîne.

Le CRV-HQ protège donc un peu les utilisateurs des complexités du langage de requête, mais peut-être pas autant qu'il serait souhaitable. Étant très proche de lui, il harnache beaucoup de son expressivité, mais laisse à l'usager une bonne partie du fardeau de comprendre le fonctionnement de SPARQL. Quoi que les utilisateurs de SQL puissent remplir plus facilement l'écart entre les deux langages étant donné la proximité de ceux-ci, des usagers néophytes de ces deux langages seront confrontés à une courbe d'apprentissage relativement élevée.

Comme on a pu le voir, plusieurs fonctionnalités peuvent venir aider à aplatir cette courbe, mais le problème réside tout de même dans la nature même des CRV : ils existent pour aider l'utilisateur à écrire une requête et non à accomplir le but qui les poussent à composer cette requête.

OM peut être considéré comme une tentative de compromis entre l'expressivité d'un CRV et l'utilisabilité d'un navigateur sémantique. Dans OM, les utilisateurs débutants sont particulièrement protégés contre la complexité du langage de requête. Quoi que la maîtrise de l'outil présente tout de même une courbe d'apprentissage, la présence des technologies du WS est en grande partie invisible aux usagers.

La première couche de protection réside dans l'utilisation de paradigmes de visualisation et d'interaction WIMP simples et répandus. La grille est un paradigme de visualisation très populaire dont l'utilisation est connue de tous les utilisateurs. Lors des présentations effectuées aux clients ceux-ci ont comparés OM à un Excel pour les graphes. La maturité des outils n'étant pas du tout la même, la comparaison est flatteuse pour OM, mais tend à montrer qu'OM garde les usagers en terrain connu.

La représentation en formulaire est un autre mécanisme d'interaction connu et intuitif d'utilisation pour les usagers. Elle permet de donner un autre point de vue sur les données en

graphe et montre, tout comme la grille, en temps réel, les conséquences des modifications demandées par l'usager.

Les super-utilisateurs d'OM, ayant accès à toutes les fonctionnalités de l'outil, peuvent être des utilisateurs débutants, intermédiaires ou experts. Quoi qu'alors moins protégés de la forme en graphe des données, il n'en reste pas moins qu'ils ont accès à des outils simples et intuitifs.

L'utilisation d'une représentation en graphe et d'une représentation en grille dont les modifications de l'une se répercutent dans l'autre, permet à l'usager de se représenter mentalement la requête qu'il a formulée de plusieurs façons. Des mécanismes simples lui permettent ainsi d'explorer à la fois le modèle et les données. De plus, la double représentation de la vue lui permet de comprendre plus facilement les interactions entre les cellules des différentes classes de la grille. Dans le cas des requêtes complexes, la représentation graphique est particulièrement utile pour concevoir rapidement la forme que prend la requête. En effet, dans certains cas, différentes requêtes mènent à une représentation très similaires dans une grille en deux dimensions. Dans ces cas, la visualisation graphique permet de distinguer la structure de la requête en un simple coup d'œil.

Les utilisateurs experts trouveront aussi leur compte avec OM. L'outil offre une grande partie de l'expressivité de SPARQL dans une interface qui soit pratique autant pour l'exploration que pour l'exploitation de l'ontologie. Cependant, des fonctionnalités semblables à celles développées dans le CRV-HQ manquent toujours afin de permettre aux utilisateurs experts des fonctionnalités plus avancées telles que de pouvoir modifier manuellement la requête sous-jacente de la vue.

Tel que mentionné précédemment, l'utilisabilité d'OM provient principalement de l'utilisation de paradigmes de visualisation répandus et appréciés, le principal étant une grille en deux dimensions contenant un graphe de données. L'hypothèse posée est que cette projection du graphe dans une matrice en deux dimensions rend la compréhension et la comparaison des données plus intuitives. Chez HQ, comme dans plusieurs entreprises, plusieurs jeux de données qui devraient être représentés en graphe sont contenus dans des tableurs tels qu'Excel. Des gammes de maintenances, des nomenclatures, des topologies, des procédures, des cartographies de connaissances, des feuilles de routes sont autant d'exemples de jeux de

données trouvant avantage à être stockés sous forme de graphes. Les tableurs n'étant pas conçus pour des données en graphe, la création, l'utilisation et la maintenance de ces jeux de données sont complexes, chronophages et sensibles aux erreurs. C'est une des raisons pour laquelle les entreprises ont besoin d'un Excel pour les graphes qui soit facile d'utilisation et intuitif. L'utilisation d'un standard tel qu'OWL facilite non seulement la création d'un tel tableur pour graphe, il entraîne également dans son sillage plusieurs avantages et possibilités.

6.3.4 Ingénierie dirigée par les ontologies

Le type de développement des applications de cette recherche peut être considéré comme de l'IDO, puisqu'un effort considérable est fait pour laisser le plus possible les composantes de l'application être guidées par l'ontologie. Ce principe directeur a mené dans DATE à l'utilisation de propriétés de visualisation servant à annoter l'ontologie de domaine pour diriger le UI. Ce procédé a atteint une forme mature dans OM par le regroupement de toute une panoplie de ces propriétés dans l'OV sous une structure facilitant leur usage dans les requêtes génériques et permettant leur évolutivité.

Alors que dans DATE, les types des champs de formulaires sont guidés par une propriété de visualisation, dans OM, ils seront déduits de l'ontologie de domaine, plus particulièrement des codomaines et des types des propriétés, ces derniers renseignant sur les cardinalités. Cet exemple montre le mouvement d'une évolution du procédé où les informations pouvant être déduites directement du modèle de données auront préséance sur l'utilisation de traductions de celles-ci même lorsque ces dernières peuvent être plus pratiques. Cet effort a comme but l'augmentation de l'universalité de l'approche en tentant de se conformer le plus possible aux standards de modélisation du W3C.

Ainsi, afin que le cadre de développement puisse le plus possible s'adapter aux connaissances déjà présentes dans les ontologies de domaines et pour qu'il puisse tirer profit de toute l'expressivité de RDF, RDFS, OWL, un effort a été fait pour implémenter l'usage des sémantiques de ces langages. En effet, une application sémantique complètement générique serait idéalement capable de tirer parti de toutes les sémantiques des langages RDF, RDFS et

OWL tout en pouvant comprendre toute ontologie utilisant un ou plusieurs de ces langages. Cependant, cette situation idéale n'est pas complètement réalisable.

D'une part, comme ces langages ontologiques n'imposent aucune contrainte sur la façon dont leurs sémantiques sont utilisées, différentes ontologies peuvent en faire usage différemment. De plus, les ontologistes peuvent introduire de nouvelles sémantiques pour modéliser l'information. Pour s'assurer d'un fonctionnement constant des interfaces il est donc nécessaire de choisir comment les sémantiques sont comprises par le système et ainsi imposer une forme minimale aux ontologies utilisables.

D'autre part, le développement étant fait en fonction des besoins, il semble utopique de vouloir faire usage d'entrée de jeu de toutes les sémantiques de ces langages. Un sous-ensemble de ces sémantiques a donc été sélectionné. Cette sélection devait être assez complète pour permettre la création d'applications démontrant des possibilités clés de l'univers sémantique et assez restreinte pour que les premières itérations de conception ne soient pas inutilement ralenties.

Les sémantiques de RDF, RDFS et OWL utilisées dans OM sont : *rdf:type*, *rdf:langString*, *rdfs:comment*, *rdfs:domain*, *rdfs:label*, *rdfs:range*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *owl:Class*, *owl:DatatypeProperty*, *owl:FunctionalProperty*, *owl:inverseOf* et *owl:ObjectPro*perty.

Cela représente environ 12,5% (2/16) de RDF, 40% (6/15) de RDFS, et 12,5% (5/40) d'OWL. Il reste donc beaucoup de place à l'évolution et l'amélioration. De plus, d'autres ontologies de haut niveau existent telles que *Simple Knowledge Organiszation System* (SKOS) servant à modéliser, partager et lier les systèmes d'organisation des connaissances ou *XML Schema Datatypes* (XSD) servant à lier des types de données XML aux données en format RDF, pour ne nommer que celles-ci. Les sémantiques de telles ontologies sont omniprésentes et leurs sémantiques devraient aussi être prises en charge par un outil tel qu'OM.

De plus, afin de garder le fonctionnement de l'outil le plus près possible des technologies du WS, il devrait être envisagé de tabler sur des implémentations conservant les structures de données préconisées par la logique de premier ordre ainsi que les standards énoncés par le W3C.

Un premier exemple d'alignement potentiel de ces standards et d'OM est observable dans le mécanisme de sauvegarde de vues. Alors que dans OM les vues sont sauvegardées comme des graphes de triplets contenant des sémantiques de l'OV, elles auraient pu être sauvegardées comme des classes avec restrictions. Cette mécanique d'OWL utilise, entre autres, les sémantiques *rdfs:subClassOf*, *owl:Restriction*, *owl:onProperty*, *owl:hasValue*, *owl:allValuesFrom*, *owl:someValuesFrom*, *owl:intersectionOf*, *owl:minCardinality* et *owl:maxCardinality*. Tel qu'il a été testé lors de cette recherche, toutes les vues sauvegardées auraient ainsi pu prendre une forme permettant de déterminer leur individu non plus par des requêtes SPARQL, mais bien par l'inférence.

Puisque l'engin d'inférence du module *Spatial and Graph* d'Oracle n'implémente qu'un sous-ensemble des sémantiques d'OWL et que *owl:Restriction*, la pierre angulaire de ce mécanisme, n'en fait pas partie, cette stratégie a donc dû être abandonnée. Elle devrait tout de même être explorée dans le futur, pour les raisons mentionnées plus haut.

Un deuxième exemple concerne la formulation des énumérations qui s'est fait dans OM par l'utilisation d'un EBox afin de pouvoir réutiliser les fonctionnalités déjà existantes pour en faire l'édition. La formulation préconisée par le W3C est foncièrement différente de celle déployée dans OM et des efforts devraient être fournis pour trouver un meilleur compromis entre les deux approches.

Dans le même ordre d'idée, l'OV devrait être conçue et son évolution devrait se faire dans une approche communautaire comme celle du WS. Il serait important d'obtenir une OV dont les standards sont partagés, améliorés et soutenus par la communauté et le W3C. Ainsi, les ontologistes pourraient développer des ontologies avec ces standards et les développeurs pourraient développer des composantes logicielles capables de les utiliser. Les ontologies et les données de n'importe quels domaines pourraient ainsi être consommées par des applications génériques et auto-adaptatives telles que celles produites grâce à OM.

6.3.5 Industrie 4.0

OM vise à occuper une niche unique dans la boîte à outils des entreprises de l'Industrie 4.0 par sa capitalisation d'avantages des technologies du WS.

Dans un contexte comme l'Industrie 4.0, où il serait intéressant que l'union entre des modèles de données puisse se faire de manière *ad hoc*, les inférences se basant sur la logique de premier ordre seraient bénéfiques. Les applications auraient alors la possibilité de s'en servir, par exemple, pour déduire automatiquement l'appartenance d'un appareil à une classe d'appareils, pour déterminer des correspondances entre des individus provenant de modèles de données différents ou encore pour tester la validité logique des valeurs.

L'approche de la méthodologie présentée ici pourrait être utilisée pour permettre le croisement *ad hoc* de données provenant de modèles de différentes sources, telles que des entreprises, des machines industrielles ou des appareils commerciaux. La généralité et l'auto-adaptivité d'OM pourrait aussi aider les applications à s'adapter aux constantes réingénieries de systèmes.

L'approche d'OM met entre les mains de l'utilisateur final le pouvoir de créer les vues qu'il désire par l'exploration *ad hoc* des données du domaine. Cette approche est pertinente dans le contexte de l'Industrie 4.0 où la recherche de flexibilité et de réactivité tend à ramener une partie du pouvoir décisionnel chez les experts de domaine. Dans une vision plus cyber-physique, ces croisements de données *ad hoc* seraient effectués par les machines.

Dans le contexte de l'Industrie 4.0, le développement d'une OV standardisée permettant d'annoter les modèles de données des appareils de la chaîne de valeur pourrait apporter des bénéfices. Par exemple, un nouvel appareil s'ajoutant à la chaîne pourrait directement être connecté aux tableaux de bord si son modèle de données est annoté avec l'OV. Ainsi, les modèles pourraient contenir l'information nécessaire pour que les applications auto-adaptives puissent fournir des fonctionnalités complexes adaptées à ces appareils.

6.4 Leçons apprises

Ces trois applications ont permis de tirer des leçons sur l'utilisation des technologies du WS dans l'implémentation d'applications Web auto-adaptatives.

Parmi les leçons apprises lors du développement de DATE, la façon d'atteindre l'indépendance conceptuelle fut sans doute la plus importante. Le développement du CRV-HQ, quant à lui, a permis de démontrer qu'il est possible de transposer les fonctionnalités des CRV SQL dans des CRV SPARQL ce qui a mis en évidence le besoin de développer des fonctionnalités possibles strictement dans le monde sémantique.

Finalement, OM a permis d'apprendre plusieurs leçons sur le développement de générateurs d'applications auto-adaptatives. Celles-ci comprennent l'utilisation de l'outil pour générer des applications, l'utilisation du chargement paresseux aux différents grains ontologiques et l'utilisation de l'approche sur différents points de terminaison en ligne. De plus, la recherche a permis d'émettre un jeu d'obligations de modélisation pour permettre à une ontologie d'être utilisée avec OM ainsi que deux jeux de recommandations respectivement pour utiliser OM de façon optimale et pour reproduire l'approche avec d'autres technologies. Finalement, cette application a permis de mettre en évidence une synergie remarquable entre différents aspects de l'approche.

6.4.1 Indépendance conceptuelle

Lors de l'implémentation de DATE, il a été établi que plusieurs des propriétés nécessaires à l'obtention de l'indépendance conceptuelle sont inhérentes aux technologies RDF. Ne coder en dur que les sémantiques des standards RDF, RDFS et OWL et laisser toutes les autres ressources des patrons de triplets codées en mou dans les requêtes SPARQL est la clé pour concevoir des requêtes génériques. En effet, puisque les sémantiques de ces trois langages sont partagées à travers toutes les ontologies basées sur le RDF, ils forment une base conceptuelle commune à tous les domaines qu'elles représentent. En limitant la dépendance conceptuelle à

ces sémantiques, les applications ainsi développées peuvent utiliser n'importe laquelle de ces ontologies, peu importe le domaine qu'elles décrivent.

6.4.2 Fonctionnalités strictement sémantiques

Le CRV-HQ a permis de démontrer la faisabilité de reproduire plusieurs des fonctionnalités du monde relationnel dans le monde sémantique. Ainsi, on peut fabriquer des solutions semblables pour des jeux de données provenant des deux mondes. Cela dit, afin de démontrer l'utilité des ontologies et d'harnacher la puissance des technologies du WS, il convient d'essayer de concevoir des fonctionnalités qui ne peuvent fonctionner strictement que dans le monde sémantique.

Le CRV-HQ présente de telles fonctionnalités. Elles concernent la nature ontologique du modèle de données. Par exemple, l'usager pourra afficher le type le plus spécialisé d'un individu à partir d'une classe plus haute dans la hiérarchie. Cette fonctionnalité est présente dans le CRV-HQ par l'entremise du bouton radio 'Afficher type' du menu contextuel des options sur les classes. Un autre exemple est la possibilité de spécialiser les classes de l'outil graphique grâce aux relations de subsomption.

Le potentiel pour de telles fonctionnalités reste cependant presque inexploité. Les possibilités offertes par l'auto-adaptabilité, l'inférence, la nature en graphe des données et la nature ontologique des modèles sont nombreuses et il est a espérer que de futures recherches permettront de les harnacher.

6.4.3 Méthodologie de création d'applications avec OM

Le CHAPITRE 5 présentait l'application PPAL, conçue grâce à OM pour des chercheurs en sciences des matériaux de l'IREQ et mise en production dans l'intranet d'HQ. Cependant, d'autres applications ont été construites en utilisant le cadre de développement OM. Il s'agit tout d'abord de DATE-2, une deuxième version de l'application DATE présentée au CHAPITRE 3, grâce au cadre de développement d'OM. Deuxièmement, Gaz Dissouts, une

application utilisant les données d'analyses chimiques de l'huile des transformateurs de puissance pour implémenter les célèbres triangles et pentagones de Duval développés à l'IREQ et utilisés dans plusieurs utilités électriques du monde pour établir l'état de santé des transformateurs.

Ces deux implémentations sont restées au stade de prototypes de démonstration. Cependant, elles ont permis de démontrer la rapidité avec laquelle OM pouvait générer des applications. Dans les deux cas, la création de l'ontologie et l'instanciation d'OM avec ce TBox et un ABox a permis d'avoir une première version de l'application désirée.

Dans une entreprise de l'envergure d'HQ, le besoin de systèmes d'information est immense. OM se veut tout d'abord utile pour le développement rapide d'applications.

La construction d'une application grâce à OM se fait en trois étapes :

1) Créer le modèle de données,
2) Importer les données dans le format OWL,
3) Créer et partager des vues.

Dans les différentes applications construites avec OM jusqu'à présent, la construction du modèle de données s'est faite à partir des données déjà existantes et des besoins de recherche à venir. Cependant, plusieurs ontologies OWL populaires fonctionnent avec OM, telles que DBpedia, GoodRelations, IEC/CIM, etc. et pourraient donc servir de base à la création d'applications.

Tel que mentionné précédemment, à la suite de la création d'une version préliminaire du modèle de données, des rencontres avec les parties prenantes permettent d'ajuster la structure et la nomenclature du modèle. La nature auto-adaptive d'OM permet de faire ces changements à n'importe quelle étape du développement sans nécessiter de maintenance. Il s'agit en quelques sortes d'un prototypage évolutif perpétuel.

L'importation des données dépend du format et de l'organisation de la source des données. Elle peut consister, par exemple, à organiser des données en format CSV et à les importer dans la BDT grâce à des utilitaires tels que Cellfie (Cellfie, 2020), un plugin de Protégé (Protégé,

2020). En ce qui concerne les projets DATE-2 et Gaz Dissouts, la phase d'importation a consisté à programmer un connecteur établissant le lien entre une BDR et la BDT d'OM.

Finalement, l'outil est prêt à être remis entre les mains des experts de domaine qui fabriqueront eux-mêmes les vues destinées à être utilisées par les autres usagers.

6.4.4 Chargement paresseux

La mise à l'échelle d'OM provient principalement de l'application du principe de chargement paresseux à un modèle ontologique. L'unité ontologique de base qui a été choisie pour OM est la classe OWL. L'espace de solution est donc construit classe par classe. En effet, chaque classe du graphe ainsi que ses propriétés et un sous-ensemble de ses individus sont ramenés indépendamment des données homologues provenant des autres classes. L'espace de solution désiré est ensuite reconstruit par le client grâce à la technologie des URI. Cette façon de procéder assure un temps pratiquement linéaire pour retrouver et afficher les données en fonction du nombre de classes impliquées dans l'espace de solution (voir section Complexité d'OM).

Le cadre de développement offre donc un chargement paresseux de la grille à deux niveaux. Tout d'abord, il est fait au niveau de la classe, chaque classe étant chargée indépendamment les unes des autres. Ensuite, il s'effectue au niveau des individus, ceux-ci étant ramenés par paquets dans le client pour faciliter leur chargement.

La suite logique serait de tenter d'instaurer un mécanisme de chargement paresseux aux niveaux des propriétés afin de rendre la complexité du cadre indépendante des ontologies. Une telle étude n'a pas été effectuée, mais des mécanismes téléchargeant un certain nombre de propriétés à la fois, jumelés avec des outils UI pour permettre à l'utilisateur de choisir parmi celles d'intérêts, seraient envisageables. Ce troisième niveau de chargement paresseux pourrait ainsi garantir la mise à l'échelle de l'application peu importe la taille et la forme de l'ontologie et du jeu de données.

6.4.5 Utilisation avec des points de terminaison SPARQL

L'approche d'OM pourrait être utilisée avec des points de terminaison SPARQL au lieu d'une BDT. Dans l'état actuel de l'application, la coopération des gestionnaires des points de terminaison serait requise car ceux-ci devraient tout d'abord utiliser l'ensemble de pré-inférence sur leurs données. De plus, ceux-ci devraient permettre aux usagers d'ajouter des éléments de l'OV afin d'établir une stratégie d'approvisionnement par la foule (*crowdsourcing*) des métas-informations.

Cela dit, l'utilisation de points de terminaison, combinée avec l'approche du chargement paresseux par classe, permettrait de changer de point de terminaison à chaque classe et donc de croiser des données provenant de multiples sources dans une même grille de manière *ad hoc*. Ceci semble une avenue hautement prometteuse pour permettre l'exploration du WS.

6.4.6 Obligations de modélisation

La généralité d'OM est obtenue en tirant profit des caractéristiques inhérentes des standards RDF, RDFS et OWL. En effet, comme toutes les requêtes qu'il effectue sur la base de triplets ne contiennent que des sémantiques de ces standards codées en dur, le reste n'étant uniquement que des variables (SPARQL ou Java), OM permet de travailler avec n'importe quelle ontologie OWL respectant quelques principes de bases. Ces principes découlent de la façon dont l'outil a été implémenté et peuvent être vu comme des limitations à l'universalité d'OM. Ces principes de bases de construction ontologique nécessaires au bon fonctionnement d'OM sont présentés au Tableau 6.16. Afin de pouvoir utiliser ce cadre de développement, les ontologies de domaines doivent se conformer à ces trois règles de modélisations.

Tableau 6.16 Obligations de modélisation

1	Chaque classe, propriété, individu et vue doit avoir absolument une et une seule étiquette dans une même langue.
2	Toutes les propriétés de l'ontologie doivent posséder un et un seul *rdfs:domain* avant inférence.
3	Toutes les propriétés de l'ontologie doivent posséder un et un seul *rdfs:range* avant inférence

Ces trois obligations pourraient ne plus être nécessaires par l'amélioration d'OM grâce à l'ajout de fonctionnalités automatiques et/ou semi-automatiques. La majorité des ontologies étudiées pendant la réalisation de cette recherche remplissait déjà ces obligations et il n'a ainsi pas été nécessaire d'adapter OM. Cependant, pour que l'outil atteigne l'universalité à laquelle il aspire, des recherches plus poussées pour trouver des solutions logicielles simples et élégantes seraient requises. Si toutefois une ou plusieurs de ces obligations ne pouvaient être retirées pour une raison ou une autre, il conviendrait alors d'expliquer à la communauté des ontologistes que l'utilisation d'une telle méthodologie nécessite l'application de standards lors de la modélisation.

La première obligation stipule qu'une ontologie doit présenter une et une seule étiquette pour chacune de ses classes, propriétés, individus et vues. Si une de ces étiquettes n'est pas présente, c'est l'URI de cette entité qui sera présenté, ces derniers n'étant pas très pratiques pour la compréhension humaine. La présence de plus d'une étiquette pour une même entité mènera à la sélection aléatoire de l'une de celles-ci par l'engin de requête de la BDT.

Les deux autres obligations stipulent que chaque propriété de l'ontologie doit avoir un et un seul domaine et un et un seul codomaine avant inférence. OM n'a pas été programmé pour fonctionner avec des ontologies ne respectant pas cette exigence et le non-respect de celle-ci pourra mener à un fonctionnement erratique du système.

6.4.7 Recommandations de modélisation

Il existe deux principes non nécessaires mais pouvant aider à l'utilisabilité ou la mise à l'échelle d'OM. Afin de tirer le maximum d'OM, il est recommandé de suivre ces recommandations présentées au Tableau 6.17.

Tableau 6.17 Recommandations de modélisation

1	Minimiser les jointures des vues sur lesquelles les opérations CRUD sont effectuées.
2	Séparer les propriétés des classes qui en contiennent plus d'une dizaine dans des classes liées à celles-ci par des propriétés de type objets fonctionnelles et inversement fonctionnelles.

La première recommandation indique que l'édition des données de la BDT pourra être simplifiée par la minimisation des jointures entre la classe pivot de la requête et les classes dont on veut permettre l'édition des données.

Tel que mentionné précédemment, la grande flexibilité d'OM permet de présenter n'importe quel modèle utilisant les principes de base de modélisation OWL. Cependant, les vues qu'il est possible de créer à partir de ces modèles dépendent des modèles eux-mêmes. Il est possible de construire une vue contenant autant de jointures que désirées, toutefois, cela implique que l'usager devra être en mesure de modifier tous les individus servant aux jointures, ce qui, dans certains cas d'utilisation particuliers, est peu pratique.

De tels cas peuvent être résolus par l'ajout d'une propriété unique liant la première et la dernière classe des jointures dans l'ontologie. De cette manière, il est possible de modifier reformuler une vue en remplaçant n'importe quel nombre de jointures par une seule. Cette fonctionnalité peut être automatisée directement par inférence ou encore par l'utilisation d'un chemin de propriétés (*property path*) un mécanisme de SPARQL pour remplacer de multiples jointures par une expression unique prenant la place d'un prédicat dans un patron de triplets. Cela dit, la façon la plus simple d'implémenter une telle propriété reste l'utilisation des outils d'édition du modèle pour créer ce lien direct entre les classes afin de constituer des vues pratiques pour ces cas d'utilisation.

En effet, les technologies du WS possèdent l'heureuse capacité de pouvoir admettre des ponts entre n'importe quelles classes, les BDT n'étant pas soumises aux notions de normalisation des BDR. Les ontologistes ont ainsi toute la latitude pour concevoir de telles propriétés qui simplifieront l'utilisation d'OM. À terme, quand OM sera pourvu d'outils d'édition de modèle, les super-utilisateurs pourront définir ces relations pendant la construction de leur vue grâce à des composantes d'interactions simples.

Pour l'instant, lorsque l'ontologiste crée un modèle pour le développement d'une application, il peut tout simplement le concevoir en fonction des cas d'utilisation pré-identifiés. Dans cette optique, il est donc recommandé de minimiser les jointures des vues sur lesquelles les opérations CRUD seront effectuées. Les opérations de consultation, de tri et de filtrage s'utilisent quant à elles de façon tout aussi intuitive, peu importe le nombre de jointures.

La deuxième recommandation concerne la mise à l'échelle de l'application. Afin de pouvoir garantir un temps d'affichage constant des données peu importe le nombre de classes, de propriétés et d'individus de l'ontologie, on doit pouvoir contrôler le grain du chargement paresseux à ces trois niveaux. Le niveau de la classe et le niveau de l'individu étant déjà pris en charge, uniquement le niveau des propriétés reste problématique. Celui-ci pourra éventuellement être résolu par des solutions automatiques ou semi-automatiques, mais dans l'état actuel d'OM, il est recommandé de limiter les propriétés d'une classe au nombre d'environ dix. Cela a le double avantage de limiter les chances de surcharge informationnelle d'une part et de limiter le temps de chargement d'autre part.

Lors de la modélisation de l'ontologie, il est souvent possible de regrouper les différentes propriétés décrivant un individu dans des classes distinctes, ces dernières étant liées à la classe désirée par une propriété fonctionnelle et inversement fonctionnelle. Le sens de l'ontologie reste alors le même, mais les utilisateurs peuvent désormais parcourir les propriétés des classes par catégories, ce qui rend l'exploration ontologique plus simple tout en accélérant le temps de réponse.

6.4.8 Recommandations pour l'approche

Afin de développer une approche semblable, il est recommandé de suivre les lignes de conception présentées au Tableau 6.18.

Tableau 6.18 Recommandations pour l'approche

1	Utiliser un engin d'inférence utilisant le chaînage arrière.
2	Utiliser un engin d'inférence permettant les classes avec restrictions sur les littéraux.
3	Encourager les gestionnaires de points de terminaison à utiliser l'ensemble de pré-inférences.
4	Encourager la communauté à utiliser une OV standardisée.

Ces recommandations regroupent des idées qui ont toutes été discutées précédemment. Elles ne seront donc ici que brièvement réabordées. Les deux premières concernent le choix de l'engin d'inférence, et les deux dernières concernent le développement communautaire de ce type d'outils.

En ce qui concerne le choix de l'engin d'inférence, dans la présente recherche l'utilisation d'un engin d'inférence ne permettant pas le chaînage arrière a obligé l'utilisation d'artifices de programmation. Dans le même ordre d'idée, l'utilisation d'un engin d'inférence capable d'utiliser toutes les sémantiques de RDFS et OWL semble de mise. Les sémantiques de classes avec restrictions permettraient une implémentation correspondant directement aux standards du WS.

Dans l'optique où l'on voudrait populariser l'utilisation de la méthodologie, il faudrait pouvoir réconcilier les paradigmes OO et de classification des données provenant des points de terminaison SPARQL à l'extérieur de l'entreprise. Comme il a été vu, une solution existe sans modification des données externes, mais il serait aussi envisageable de sensibiliser les gestionnaires de ces points de terminaison et de les encourager à utiliser le jeu de pré-inférences sur leurs propres données.

Pour permettre la démocratisation de la méthodologie, on pourrait aussi encourager la communauté du WS à l'utilisation de l'OV et tenter de démarrer un groupe de réflexion sur la standardisation de celle-ci par le W3C. Cette OV devrait entre autres permettre d'assurer la provenance, la qualité et l'imputabilité des données. Pour ce faire, un mécanisme utilisant une série de méta-propriétés ayant comme domaine *owl:Thing* comme c'est le cas dans OM pourrait être envisagé.

6.5 Conclusion

Plusieurs discussions sur les fonctionnalités, les enjeux techniques, les réflexions théoriques et les leçons apprises durant la conception, le développement et l'utilisation des trois applications présentées dans cette recherche, viennent d'être présentées.

De ces discussions émane la conclusion que cette approche pour fabriquer des applications en utilisant les technologies du WS permet de mettre en place une remarquable synergie s'opérant sur plusieurs niveaux.

La première intuition de cette synergie provient du constat de base que l'indépendance conceptuelle, par les capacités d'auto-adaptabilité des composantes logicielles, entraîne une

généricité permettant de créer de nouvelles applications. On fait alors d'une pierre deux coups en diminuant à la fois les coûts de maintenance et de développement des applications.

S'ajoute à cette synergie le fait que ces applications, en pouvant s'auto-adapter, permettent *de facto* le prototypage évolutif perpétuel. Ce concept est en phase avec les méthodologies agiles où l'on voudra raffiner l'application et son modèle de données à chaque itération. Ces modifications n'entraînant plus la réécriture du code, ce qui vient encore augmenter les économies liées au développement.

L'utilisation des technologies du WS qui permettent l'implémentation simple de cette indépendance conceptuelle entraîne dans son sillage d'autres propriétés souhaitables pour les applications. Les requêtes génériques utilisées systématiquement pour interroger la base de connaissance permettent de ramener les données par des chargements paresseux sur trois niveaux ontologiques : la classe, la propriété et l'individu.

Théoriquement, ces chargements paresseux assureront non seulement un temps constant de visualisation des informations peu importe la taille de l'ontologie ou du jeu de données, mais permettront de croiser de manière *ad hoc* dans une seule et même grille les données provenant de sources distinctes. Le nuage de données ouvertes et liées comportant à lui seul 1255 jeux de données en 2020, les possibilités impliquées par un tel outil sont immenses. Et celui-ci est manœuvrable par des utilisateurs néophytes de ces technologies.

De surcroît, les capacités de raisonnement automatisé des moteurs d'inférence pourront éventuellement être harnachées pour enrichir ces données par l'information implicite émanant non seulement des jeux de données individuels mais de leur croisement.

CONCLUSION

La mise en production de PPAL, une application construite grâce au cadre de développement OM axé sur l'auto-adaptabilité par l'utilisation des technologies du WS, dans les systèmes d'entreprises d'HQ démontre la faisabilité de l'approche. Cette méthodologie favorise la généralité, l'universalité et l'utilisabilité des applications générées et de multiples synergies en émergent.

Voici maintenant les conclusions qu'on peut tirer de chacune des applications développées pendant cette recherche.

7.1 Données d'analyse des transformateurs électriques

Les résultats attendus de la première étape étaient doubles. Tout d'abord, il était nécessaire de prouver qu'une application auto-adaptative était faisable sous la forme désirée et, pour ce faire, il fallait développer un véhicule de transfert d'information qui soit générique pour que le côté serveur et le côté client puissent communiquer dans ce contexte. Une preuve que l'auto-adaptabilité est possible grâce aux propriétés intrinsèques des BDT a été obtenue par la mise en œuvre de cette première application.

Avec l'utilisation d'une représentation RDF pour stocker l'information, des requêtes SPARQL génériques qui peuvent chercher dans n'importe quel graphe sémantique à la fois pour des connaissances conceptuelles et pour des individus ont pu être développées. Elles apportent au système la capacité de s'adapter à l'évolution du modèle conceptuel et lui permettent d'être utilisé pour différents domaines de savoir.

Cette application, DATE, permet le suivi de l'état de santé des transformateurs de puissance en fonction de la présence de composés chimiques dans leur huile. DATE entre dans la catégorie des navigateurs sémantiques et permet l'exploration ontologique par les individus. On pourrait catégoriser plus précisément cette application comme étant un navigateur sémantique textuel spécialisé pour la présentation de données sous forme de grilles. Ce concept sera poussé à sa conclusion logique dans la troisième application : OM.

DATE permet principalement des fonctions de création, de lecture, de mise à jour et de suppression de données. Une revue des différentes fonctionnalités des navigateurs sémantiques textuels a permis d'identifier des pistes de développement pour DATE qui ont ultimement menées aux applications subséquentes.

De même, l'étude des fonctionnalités d'une telle application pour son utilisation par des utilisateurs débutants en se basant sur les travaux de Dadzie et Rowe (2011) ont permis de conceptualiser des pistes d'amélioration.

Cette application terminée, un article de conférence (Bhérer, Vouligny, Gaha, Redouane, & Desrosiers, 2015) a été rédigé sur les principes mis en œuvre. Celui-ci a été présenté à la conférence SÉMAPR0 2015 portant sur les avancées dans le domaine du traitement sémantique et a été sélectionné sur la base de son contenu pour faire l'objet d'une version étendue afin d'être publié dans l'International Journal On Advances in Intelligent Systems (Bhérer, Vouligny, Gaha, & Desrosiers, 2016).

La version étendue introduit la comparaison entre les deux types de véhicules génériques présentés au CHAPITRE 3. La supériorité du véhicule générique conservant la structure en graphe des données ne s'était alors pas encore imposée de façon aussi marquante qu'après avoir développé les deux autres applications. Les avantages d'un tel véhicule sont ses capacités d'évolutivité automatique, l'utilisation potentielle des capacités d'inférence en local et sa double utilisation pour le transfert de l'information et pour la sauvegarde des vues. Ce dernier avantage permet de faciliter encore davantage l'évolution des applications en permettant le développement de mécanismes automatiques pour détecter les conséquences des changements apportés à l'ontologie sur les vues créées par les utilisateurs.

Tel que proposé dans ces articles, un système d'information auto-adaptatif basé sur une BDT est plus simple à implémenter qu'un système d'information auto-adaptatif utilisant du XML pour dynamiser des fonctions sur une BDR, comme dans les travaux de McGinnes et Kapros (McGinnes & Kapros, 2015). Plusieurs artifices qui doivent être considérés lorsqu'on bâti une application auto-adaptative depuis une BDR, ne sont plus nécessaires grâce à l'utilisation des technologies du WS, telle que décrit dans le Tableau 1.1. L'utilisation d'une librairie pour

cartographier une BDR dans une BDT apparaît alors judicieuse afin de générer l'indépendance conceptuelle nécessaire à une telle application.

DATE a permis de mettre en lumière la similarité des objectifs de l'IDM et l'IDO avec ceux de la quête de l'indépendance conceptuelle. En outre, DATE a permis de développer une version préliminaire de ce qui deviendra dans OM une ontologie de visualisation en bonne et due forme.

Bien que les possibilités d'exploration ontologique de DATE fussent limitées, sa réalisation a permis de démontrer qu'une application auto-adaptative peut atteindre l'indépendance conceptuelle en capitalisant sur les technologies RDF et ses propriétés inhérentes. De tels systèmes pourraient être avantageux à utiliser dans des domaines où l'évolution des connaissances est rapide. Ils s'inscrivent bien dans une philosophie de développement agile, permettant au modèle de données d'évoluer librement à chaque itération. Ces considérations ont permis de croire que des applications auto-adaptatives pourraient apporter des réductions de coût substantielles au développement applicatif et à la maintenance et ont motivées la poursuite des travaux de recherche.

7.2 Constructeur de requêtes visuelles SPARQL d'Hydro-Québec

La deuxième application réalisée dans le cadre de cette recherche est un CRV se basant sur les paradigmes visuels présents dans les outils relationnels les plus populaires qui allient la représentation en graphe, la représentation en formulaire et la représentation par facettes. Celui-ci a été développé pour permettre aux ingénieurs d'HQ d'explorer et de faire des requêtes *ad hoc* sur les BDT de l'entreprise sans toutefois en connaître le langage de requêtes.

Dans cet outil, l'exploration ontologique est devenue générique et se fait désormais par le modèle au lieu des individus. Un mécanisme permet toutefois de prendre en compte les individus dans l'exploration du modèle. On peut ainsi permettre l'exploration uniquement des classes du modèle contenant des individus et filtrer les résultats uniquement sur des propriétés instanciées chez ces individus.

Contrairement à DATE, dans le CRV-HQ l'expressivité du langage SPARQL est presque complètement mise à la disposition des utilisateurs, ce qui vient au coût de l'utilisabilité de l'outil. Le CRV-HQ permet donc aux utilisateurs de tirer profit d'un maximum de sémantiques de SPARQL, mais ne les protège que peu de la complexité de celui-ci et se rapproche d'un LRV sans toutefois franchir la limite conceptuelle vers ce type d'outil.

Cette généricité et cette expressivité ont permis le développement d'une pléthore de fonctionnalités. Ces fonctionnalités ont été comparées aux fonctionnalités du CRV relationnel Toad et aux fonctionnalités d'autres CRV sémantiques. Elles ont aussi été comparées aux fonctionnalités recommandées pour les outils d'exploration ontologique tirées de l'étude de Heim et Ziegler (2009). Finalement, elles ont été mises en relief avec les fonctionnalités recommandées pour les applications sémantiques pour débutants et celles recommandées pour les applications sémantiques pour utilisateurs avancés de l'étude de Dadzie et Rowe (2011). La faisabilité et l'effort nécessaire pour implémenter ces fonctionnalités dans le CRV-HQ ont été évalués et pourront servir à guider le développement de versions ultérieures plus évoluées.

L'idée à la base de la conception du CRV-HQ était de maximiser l'acceptabilité de l'outil par l'utilisation de principes de visualisation et de mécanismes d'interaction tirés des CRV relationnels les plus populaires, un effort qui semble absent de la littérature. Le CRV-HQ s'acquitte de la majorité des tâches issues du monde relationnel. La fonctionnalité qui fait le plus défaut au CRV-HQ par rapport au CRV de Toad, est la possibilité de visualiser l'espace de solution de la requête au fur et à mesure que l'utilisateur construit celle-ci.

À ce stade de la recherche, les possibilités inférentielles offertes par OWL commencent à être explorées. Cette utilisation n'en est encore qu'à ses balbutiements dans le CRV-HQ, mais ouvrira la porte aux expérimentations avec OM.

Le premier enjeu majeur dans le développement d'un tel outil de visualisation sémantique est l'optimisation des requêtes. Cet enjeu devrait être abordé de prime abord dans les prochaines versions de l'outil. La pagination des résultats permet d'augmenter la performance mais elle complexifie non-trivialement le temps des BDT pour résoudre les requêtes.

L'état de l'art actuel contient plusieurs approches pour obtenir un CRV SPARQL facilitant l'utilisation par des néophytes de ces technologies. Cependant, aucune d'entre elles ne

s'appuient sur l'analyse des constructeurs de requêtes visuelles SQL qui ont pourtant près de vingt ans d'existence et sont déjà au summum de leur art. Cette application fut donc réalisée en maximisant la ressemblance de l'outil développé avec ses équivalents relationnels ainsi qu'en intégrant les meilleures pratiques en œuvres dans les outils sémantiques similaires. Cet outil, fondamentalement différent des autres applications réalisées dans le cadre de cette recherche aura, s'il est adopté par les employés d'HQ et/ou par des membres de la communauté sémantique, ses propres retombées. Il permettra de démocratiser l'interrogation des bases de données sémantiques en ne sacrifiant qu'un peu de l'expressivité de SPARQL pour l'obtention d'une plus grande utilisabilité. Il pourrait en particulier être un vecteur d'adoption des technologies sémantiques en facilitant le passage des utilisateurs habitués aux technologies relationnelles grâce à l'utilisation d'une interface graphique semblable, connue, appréciée et ayant passée l'épreuve du temps.

Les principales leçons tirées de la réalisation de cet outil sont donc les suivantes :

1) On devra sacrifier une plus grande partie de l'utilisabilité pour protéger les utilisateurs néophytes de la complexité de SPARQL;
2) Maintenant qu'on sait comment imiter les fonctionnalités du monde relationnel dans le monde sémantique et afin de démontrer les avantages de l'utilisation des technologies du WS pour ce type d'application, des fonctionnalités de visualisation et d'interactions pratiquement impossibles dans le monde relationnel devraient être développées;
3) Les capacités d'inférence pourraient être davantage mises à profit;
4) L'outil gagnerait énormément en intuitivité s'il montrait l'espace des résultats au fur et à mesure que l'utilisateur construit sa requête.

Ces quatre leçons ont servi à orienter le développement de la troisième application : OM.

7.3 Owl Monkey

Le troisième outil développé est un constructeur de vues utilisant la représentation par facettes et la représentation par formulaire pour permettre la construction dynamique d'une vue sur les données. Cette vue est présentée dans un nouveau composant Web : la grille sémantique. Cette

grille sémantique permet un chargement paresseux à deux grains ontologiques, l'individu et la classe.

Une représentation graphique des requêtes tirée du CRV-HQ offre un moyen d'interaction alternatif à l'exploration ontologique et la construction de vues. OM pouvant s'utiliser avec n'importe quelle ontologie OWL, il peut être utilisé comme pièce maîtresse d'une méthodologie pour générer des applications auto-adaptatives. De telles applications ne nécessitent pas de maintenance lorsque leur modèle de données évolue, rendant donc les systèmes d'information dans lesquelles elles prennent place, plus réactifs à l'évolution de l'environnement et des connaissances.

Dans le cadre de cette recherche, OM a été utilisé pour développer une première application, PPAL, servant à l'échange et à l'exploration de données sur les matériaux autolubrifiants entre diverses parties prenantes chez HQ. PPAL a été mise en production dans les systèmes d'entreprise d'HQ.

OM peut être utilisé autant par des utilisateurs néophytes des technologies sémantiques que par des utilisateurs experts, on parlera donc ici d'utilisabilité universelle (Shneiderman & Plaisant, 2010). Cependant, son développement s'est effectué autour d'utilisateurs néophytes. En effet, l'idée ici était de remettre entre les mains des experts de domaines la possibilité de façonner des vues par l'exploration *ad hoc* de l'ontologie et des données. Cet outil sera particulièrement utile pour la représentation et la manipulation de données se présentant naturellement sous forme de graphes, telles que les nomenclatures, les gammes de maintenance, les feuilles de routes, etc.

Les utilisateurs néophytes d'OM sont protégés de la complexité de SPARQL par des mécanismes de visualisation et d'interaction connus, simples et intuitifs et par la présence de plusieurs mécanismes différents pour réaliser les mêmes tâches (deux types d'exploration, deux types de formulaires CRUD, etc.). Les utilisateurs experts, quant à eux, peuvent profiter de l'expressivité de l'outil. Finalement, l'évolutivité de l'outil permettra d'ajouter des composantes et fonctionnalités supplémentaires qui viendront augmenter cette expressivité.

OM permet de construire des applications par la réutilisation de composantes génériques et flexibles se basant elles-mêmes sur des requêtes génériques. Aucun code n'est donc généré

lors de la construction d'une nouvelle application, si ce n'est le modèle de données lui-même. Cette approche par canevas génériques destinés à être pilotés par des utilisateurs néophytes semble absente de la littérature.

Dans OM, les fonctionnalités sur les données dépendent de l'ontologie de domaine et de l'ontologie de visualisation, ce qui s'inscrit dans une perspective d'ingénierie dirigée par les ontologies. Une grande partie des sémantiques qu'offre RDF, RDFS et OWL sont encore inutilisées, ce qui offre autant d'opportunités d'évolution de l'application. L'ontologie de visualisation peut être considérée comme une contribution en soi et servirait idéalement de point de départ à la construction d'une ontologie adoptée par la communauté et servant à annoter les ontologies de domaines avec des sémantiques pouvant être traduites en fonctionnalités d'applications Web.

Dans OM, comme dans plusieurs autres outils sémantiques, l'exploration ontologique se fait par les propriétés de type objets de l'ontologie qui relient les classes entre elles. Puisqu'un graphe peut présenter des patrons cycliques, des stratégies ont dû être développées pour identifier de façon unique chaque manifestation d'une classe et d'une propriété. De plus, afin de pouvoir afficher un graphe d'un nombre indéterminé de dimensions dans une table en 2 dimensions, l'outil de visualisation et d'interaction, la grille sémantique, a dû être affublée de fonctionnalités supplémentaires. L'exploration ontologique peut se faire à partir de cette grille qui permet de visualiser l'espace de solution au fur et à mesure que l'usager le construit. Cette grille représente aussi une contribution en soi et, quoi qu'elle ne soit probablement pas encore à pleine maturité, elle permet d'atteindre plusieurs des objectifs d'OM.

Les fonctionnalités d'OM ont été comparées aux fonctionnalités d'autres outils de navigation sémantique. Elles ont aussi été analysées en fonction des défis de la visualisation des données dans des applications pour utilisateurs néophytes, des exigences de tous systèmes de visualisation, des exigences des applications sémantiques pour utilisateurs débutants et des exigences des applications sémantiques pour utilisateurs experts. Ces défis et exigences sont tirés de Dadzie et Rowe (2011), un article phare dans le domaine de la visualisation des graphes RDF. Les défis techniques liés à la représentation visuelle définies dans Liu, Cui, Wu & Liu

(2014) ont aussi été analysés dans le cadre d'OM. Plusieurs conclusions ont été tirées de ces analyses et elles permettront l'amélioration d'un tel outil dans des phases ultérieures.

Une des complexités saillantes de l'implémentation d'un outil tel qu'OM consiste à dresser des ponts entre le paradigme OO et le paradigme de classification. Cette complexité a été présentée et analysée et plusieurs pistes de solutions ont été envisagées et implémentées. La solution privilégiée par OM consiste en un jeu de pré-inférences permettant la réconciliation des deux mondes et laissant une certaine marge de manoeuvre dans son utilisation. Ce jeu de pré-inférences peut être considéré comme une contribution collatérale du développement d'OM.

Tout comme pour DATE, la généricité d'OM provient des caractéristiques inhérentes des standards RDF, RDFS et OWL. L'utilisation des standards du monde ontologique, la construction de requêtes génériques qui interrogent le méta-modèle, le modèle et les données simultanément, et l'utilisation de composantes génériques sont garantes de cette généricité. Celle-ci permet dans OM d'accélérer la création d'application et la diminution de l'effort de maintenance en entraînant des propriétés d'auto-adaptabilité. Les changements pris en compte par cette auto-adaptabilité ont été discutés et des idées pour prendre en charge automatiquement et semi-automatiquement les autres changements ont été évoquées.

Cette auto-adaptabilité provient des caractéristiques inhérentes des technologies du WS. Le véhicule générique assurant le transport des informations entre le client et le serveur, s'il garde sa forme de graphe, est aussi sujet à être auto-adaptable. Cela permet de faciliter l'évolution d'OM et de ses fonctionnalités en ne nécessitant pas la réécriture du code existant, seulement des changements à l'ontologie de visualisation et des annotations à l'ontologie de domaine.

L'approche d'OM s'inscrit bien dans le mouvement de l'Industrie 4.0 puisqu'elle ramène le pouvoir décisionnel chez les experts des domaines. De plus, l'approche pourrait s'avérer utile pour fusionner et modifier des modèles de données comme il est constamment nécessaire de faire dans le cadre des chaînes de production flexibles ou de l'internet industriel des objets. OM pourrait s'y avérer utile en permettant de croiser de manière *ad hoc* ces modèles de données. Finalement, l'utilisation des technologies du WS permettrait aux chaînes de production d'harnacher les possibilités des moteurs d'inférence.

OM peut être utilisé avec n'importe quelle ontologie respectant trois contraintes de bases. Ces obligations de modélisation ont été présentées au Tableau 6.16. De plus, des recommandations de modélisation pour permettre une plus grande performance ont été présentées au Tableau 6.17. Finalement, des recommandations pour mettre en place une méthodologie similaire à celle présentée dans cette recherche ont été présentées au Tableau 6.18.

Afin de démontrer la rapidité avec laquelle la méthodologie pouvait être utilisée, deux nouvelles applications ont été produites avec OM : une deuxième version de DATE et une application pour l'analyse des gaz dissouts dans l'huile des transformateurs de puissance. Cette méthodologie de création d'une application grâce à OM a été détaillée et consiste essentiellement à développer le modèle de données puis de l'importer dans une nouvelle instance d'OM.

La complexité des opérations d'OM a été discutée. En permettant de calibrer la granularité des différents composantes ontologiques que sont la classe, la propriété et l'individu, il est théoriquement possible d'obtenir une mise à l'échelle de la méthodologie, peu importe la quantité d'entités ontologiques contenues dans le modèle. Cela dit, l'optimisateur de requêtes d'Oracle ne permet pas de résoudre une des requêtes de la pile dans un temps raisonnable pour des classes contenant un nombre un tant soit peu élevé d'individus. Des tests doivent donc être effectués avec d'autres engins de BDT afin de pouvoir valider ces hypothèses. Ladite requête a été analysée et rien n'empêche de croire qu'un autre engin ne pourrait la résoudre dans un temps raisonnable. Cependant, si tel est le cas, afin de permettre l'essor d'une telle méthodologie, le premier focus de travaux ultérieurs devrait être le développement d'algorithme dans la BDT pour résoudre ce genre de requêtes dans un temps optimal.

L'utilisation du module *Spatial and Graph* d'Oracle pour la mise en place d'une telle méthodologie est aussi remise en cause par la non prise en charge de plusieurs sémantiques d'OWL par son engin d'inférence. L'absence de chaînage arrière est en outre un aspect qui rend l'utilisation de leur engin non optimale pour ce cas d'utilisation. Tous ces arguments ne permettent pas de recommander Oracle dans l'état actuel de leur technologie pour le développement d'outils tels qu'OM. Cela est d'autant plus malheureux que tous les gens d'Oracle consultés pendant cette recherche ont été dévoués et sympathiques.

En somme, la méthodologie présentée dans cette recherche entraîne une foule d'effets positifs. Elle offre de remarquables synergies autant au niveau de ses composantes et fonctionnalités qu'au niveau de ses principes et objectifs. Quoi que pour l'instant, elle n'ait fonctionné que sur de petites ontologies, rien n'empêche théoriquement son amélioration pour qu'elle fonctionne avec de plus grandes ontologies.

Les cinq constats suivants résument bien la force de la méthodologie et ce qui devrait être fait pour assurer son essor :

1) Le langage de méta-modélisation est la composante permanente du savoir. Les éléments de ce langage sont donc réutilisés systématiquement pour chaque savoir, peu importe la nature de celui-ci;
2) Les granularités ontologiques des connaissances peuvent se résumer en termes de classe, de propriété et d'individu. Un cadre de développement permettant un chargement paresseux sur chacun de ces niveaux permet la visualisation et l'interaction performante de n'importe quel graphe de connaissances;
3) En offrant un véhicule de transfert d'information Web générique standardisé présentant ces trois niveaux ontologiques, les développeurs d'applications ou de composantes Web, n'ont ni à comprendre ni à utiliser SPARQL;
4) Une ontologie des sémantiques de visualisation nécessaires aux fonctionnalités des applications doit être développée, partagée et standardisée afin de permettre le développement d'applications Web dirigées par les ontologies;
5) Les applications Web ainsi construites pourraient croiser les informations provenant de n'importe quel point d'accès et présenteraient des propriétés auto-adaptatives facilitant leur fabrication et leur maintenance et permettant la réutilisation de leurs composantes. Ce type de croisements *ad hoc* de n'importe quel jeu de données ouvertes et liées semble pouvoir démocratiser l'utilisation d'un Web de données tel qu'imaginé par Tim Berners-Lee.

7.4 Retombées industrielles potentielles

Les économies qu'on peut espérer d'une telle approche proviennent principalement de la diminution du temps de création et de maintenance des applications.

En effet, les capacités d'auto-adaptabilité des technologies du WS mises en pratique dans le constructeur d'applications offrent un double avantage. Tout d'abord, les applications ainsi développées s'adaptent automatiquement à plusieurs changements pouvant survenir dans le ou les modèle(s) de données, réduisant ainsi le temps et les ressources nécessaires à leur évolution. Ensuite, les composantes auto-adaptatives logicielles se basant uniquement sur des axiomes génériques de modélisation peuvent utiliser comme intrants des ontologies décrivant virtuellement n'importe quel domaine. Elles peuvent donc entrer dans la conception d'applications couvrant des domaines variés.

Le constructeur d'applications tire profit de cette généricité des composantes pour rendre la création d'applications très rapide puisqu'ainsi la seule conception du modèle de données permet l'obtention d'une application. De plus, l'effort mis sur l'utilisabilité de l'interface utilisateur permet à des experts des domaines de personnaliser eux-mêmes ces applications sans avoir à recourir à des experts des technologies de l'information, réduisant ainsi les coûts de mise en production de telles applications.

En outre, les modèles utilisés dans les applications créées grâce à OM peuvent évoluer sans que les applications ne cessent d'afficher l'information adéquate. Il en découle que les nouveaux éléments de savoir peuvent immédiatement être ajoutés et sont instantanément partagés et ce, sans avoir à passer par les canaux formels de modifications. Les processus de maintenance logicielle standards des grandes entreprises impliquent la coordination entre plusieurs départements, ce qui augmente d'autant plus les économies liées à ces processus.

Un autre avantage majeur des applications créées avec OM provient de leur auto-adaptabilité. Dans un contexte d'entreprise, l'adaptabilité aux changements provenant de la science, des marchés ou des décisions administratives est cruciale à sa réactivité globale. Une telle méthodologie devrait entraîner une augmentation de la réactivité d'une entreprise en permettant à ses systèmes d'information d'évoluer plus rapidement. On peut donc croire que l'entreprise aura une plus grande rapidité d'adaptation aux changements de son environnement.

Dans une entreprise comme HQ possédant un centre de recherche, les connaissances du domaine sont en constante évolution. Des applications auto-adaptatives pourraient dans ce contexte signifier un temps réduit d'implantation des découvertes dans les processus de

l'entreprise et cette connaissance pourrait être directement ajoutée par les experts des domaines sans avoir recours aux unités informatiques.

D'autre part, les applications développées dans cette recherche l'ont été avec un effort constant pour faciliter leur utilisation par des usagers néophytes des technologies du WS voire des technologies informatiques en général. Cela s'est fait en fournissant des fonctionnalités et paradigmes de visualisation les plus possible connus, simples et intuitifs. Un tel effort devrait entraîner dans son sillage une diminution du temps d'apprentissage des applications ainsi qu'une adoption accrue de celles-ci.

Une telle approche permet de réduire le temps investi par les experts informatiques dans le développement et la maintenance d'applications en diminuant le temps pour déterminer les besoins fonctionnels des clients et en laissant le soin aux experts des domaines de créer et modifier les vues de l'application.

En remettant entre les mains des experts de domaines le pouvoir de composer, de modifier et de gérer les vues, on leur donne une plus grande liberté ainsi que la possibilité d'explorer les données et de réorganiser l'application à leur guise. On ramène ainsi une partie du pouvoir décisionnel plus près des ressources impactées par la décision, ce qui peut aussi augmenter la réactivité de l'entreprise.

Cette auto-adaptabilité fait d'OM une avenue intéressante dans le contexte de l'Industrie 4.0 où l'on veut pouvoir croiser de façon *ad hoc* des données provenant de plusieurs modèles sans que ceux-ci ne soient connus *a priori* et que les applications ainsi créées s'adaptent aux constantes réingénieries des systèmes.

7.5 Limitations

La présente recherche présente certaines limitations et il convient maintenant de les présenter. Plusieurs de ces limites proviennent directement ou indirectement du fait qu'elle a été réalisée dans le cas d'un doctorat appliqué dans une entreprise. En effet, cette recherche a été fortement influencée par les délais et les besoins des différents clients. Ainsi, plusieurs questions théoriques, plusieurs idées et plusieurs concepts n'ont jamais pu être testés dû au manque de

temps et à la subordination de leur importance face aux priorités de livraisons. Cela dit, cet état des choses a aussi apporté une valeur ajoutée indéniable à ce travail ainsi qu'aux applications développées. Ceci est particulièrement vrai pour PPAL qui a dû passer toutes les étapes d'assurance qualité avant d'être finalement mise en production dans les systèmes d'entreprises.

Les choix des technologies ont aussi été contraints par le contexte industriel de la recherche, ce qui a mené à l'utilisation de technologies suboptimales par rapport aux buts de la recherche. Oracle en est évidemment l'exemple le plus saillant puisque cet outil présente plusieurs caractéristiques qui sont plus ou moins en contradiction avec les objectifs de la méthodologie comme l'absence de chaînage arrière, la non prise en charge de plusieurs sémantiques OWL et la présence d'un optimisateur de requête SQL peinant à traiter les requêtes traduites du SPARQL.

Une autre limitation de cette recherche provient de son caractère généraliste. En effet, des solutions informatiques complètes (*end-to-end*) ont dû être développées. Cela laisse évidemment moins de temps et d'énergie à fournir sur chacun des éléments de la recherche afin de tout d'abord s'assurer d'obtenir une solution fonctionnelle de bout en bout. De plus, cela oblige à passer du temps à trouver des solutions à plusieurs problèmes d'ordres secondaires par rapport à l'objectif principal de la recherche. Cependant, cet effort apporte aussi son lot d'innovations qui peuvent être considérées comme des contributions à cette recherche.

La principale limitation de cette recherche provient de l'absence d'une expérimentation permettant de quantifier numériquement les bénéfices de l'approche. Une telle expérimentation aurait pu être fait en comparant par exemple le temps de développement d'une même application par les méthodologies usuelles de l'entreprise d'une part et par la méthodologie de cette recherche d'une autre part. Un tel dédoublement d'effort est cependant un peu à contre-courant des pratiques en entreprise. Cela dit, l'ultime but de cette recherche aurait pu être de quantifier les économies engendrées par la méthodologie de développement d'applications sémantiques auto-adaptatives et non uniquement d'en vérifier la faisabilité et l'applicabilité dans un contexte d'entreprise. Ces économies seraient provenues principalement de la diminution du temps de création, de maintenance et d'évolution des applications.

D’autres expérimentations numériques auraient aussi pu être réalisées sur pratiquement chaque aspect des applications développées. Si ces recherches sont continuées un jour, il serait recommandé d’envisager de commencer par de tels calculs afin de s’assurer de l’optimalité de chacune des composantes et de leur apport aux gains totaux de la méthodologie.

De plus, les fonctionnalités développées dans le cadre de l’application auraient dû faire l’objet de tests d’utilisation par des usagers. Ces expériences prévues au commencement de la recherche n’ont malheureusement pas eu lieu pour diverses raisons.

Un des principaux obstacles à l’approche proposée est la performance des applications générées. Pour l’instant, toutes les applications développées dans le cadre de cette recherche n’ont pas atteint un volume de données suffisant pour causer de problèmes de fonctionnement. Cependant, l’utilisation des applications déjà développées a démontré que des requêtes SPARQL suffisamment complexes sur des jeux de données suffisamment grands provoquent des délais importants. C’est pourquoi plusieurs choix de conception ont été dirigé en fonction de l’augmentation de la performance et des possibilités de mise à l’échelle des applications.

Une autre des limitations concerne l’inférence. Comme les impacts d’utiliser le chaînage arrière sont mitigés par le déclenchement des mécanismes d’inférence hors des heures d’utilisations, il en découle que certaines des conséquences des modifications des usagers ne sont visibles que le lendemain, ce qui peut être inacceptable pour certains cas d’utilisation. Un autre moyen de mitigation serait de modéliser les ontologies en fonction des besoins des usagers du mode simple d’OM, en prenant en compte cette limitation mais ce moyen nuit évidemment à l’universalité de l’application. Ceci entraîne donc la conclusion suivante : si le chaînage arrière ne peut être effectué de façon à ne pas ralentir démesurément l’utilisation de la BDT par les utilisateurs, celui-ci n’est pas recommandé pour une utilisation telle qu’OM.

7.6 Travaux futurs

Si cette recherche trouve écho dans la communauté du WS, voici quelques pistes à envisager pour la poursuite de ces travaux.

En ce qui concerne DATE, cette preuve de concept a servi à engendrer des applications plus évoluées. Une version de DATE générée à partir d'OM a été entamée et c'est cette nouvelle version qui devrait être la cible des efforts industriels. Il serait donc recommandé de ne pas poursuivre le développement de la version originale de DATE, mais de se concentrer sur celle composée par OM. Celui-ci pourrait en effet être rapidement améliorée pour présenter toutes les fonctionnalités qui étaient présentes dans la version de DATE d'origine.

En ce qui concerne le CRV-HQ, plusieurs améliorations sont envisageables afin d'en produire une version améliorée. On peut penser aux exemples suivants :

1) Les prochaines versions pourraient aisément implémenter la possibilité d'agréger plusieurs modèles et plusieurs jeux de données;
2) Les relations de subsomptions inverses pourraient permettre de généraliser une classe en sa superclasse;
3) Des fonctionnalités pourraient permettre de remanier automatiquement l'espace de requête graphique lorsque la requête est modifiée manuellement comme le fait le CRV de Toad.

Des listes formant un ensemble beaucoup plus exhaustif des fonctionnalités qui pourraient être ajoutées au CRV-HQ sont répertoriées dans les tableaux de comparaisons des fonctionnalités.

En ce qui concerne OM, plusieurs améliorations potentielles ont été répertoriées et seront maintenant présentées. Ces pistes de travaux futurs sont séparées selon les catégories de modification de modèle, de statistiques d'utilisation, de croisement ontologique, d'évaluation des applications et du cadre, d'augmentation de la performance, d'ajout de nouvelles fonctionnalités, d'amélioration des fonctionnalités existantes, de personnalisation des applications, d'utilisation des standards du Web sémantique et d'amélioration des mécanismes d'exploration.

7.6.1 Modification de modèle

1) Ajouter des fonctionnalités d'éditions des modèles de données;

2) Étudier exhaustivement les impacts de l'édition du modèle sur les sauvegardes et développer des approches automatiques et/ou semi-automatiques pour mettre à jour les sauvegardes affectées;
3) Développer des mécanismes de notifications automatiques lors des changements aux modèles impactant les vues;
4) Permettre l'ajout des propriétés de type objets faisant le pont directement entre des classes préalablement non liées ou ajouter des propriétés de type objets étant l'équivalent de chemins de propriété au sens de SPARQL (*property path*);
5) Développer des mécanismes pour automatiquement lier certaines propriétés de l'ontologie de visualisation aux ontologies de domaines.

7.6.2 Statistiques d'utilisation

1) Produire des statistiques sur l'utilisation des différentes entités ontologiques dans les vues, dans les recherches, etc.;
2) Utiliser ces statistiques à des fins d'amélioration continue des outils ou pour proposer des modifications aux ontologies.

7.6.3 Croisement ontologique

1) Des travaux envisageables à plus long terme concernent les capacités d'OM à croiser automatiquement des individus d'ontologies différentes. Un fort potentiel de l'application se trouve dans la possibilité de permettre aux utilisateurs finaux de croiser différents modèles OWL et leurs données. Puisque chaque classe est chargée indépendamment des autres classes, on peut entrevoir la possibilité d'effectuer des requêtes génériques sur différentes BDT pour remplir une seule et même grille;
2) Afin de permettre l'utilisation d'OM avec plusieurs ontologies, il serait possible de présenter la liste des classes comme un arbre dont les branches sont les ontologies disponibles et les feuilles, des classes de ces ontologies.

7.6.4 Évaluation des applications et du cadre

1) Comparer les coûts de construction et de maintenance des applications conventionnelles à ceux des applications sémantiques auto-adaptatives générées grâce à la méthodologie. Comme une baisse de performance est attendue du fait que l'auto-adaptabilité entraîne une complexité computationnelle supplémentaire, il devrait être tenté de définir comment ce compromis se traduit dans la réalité des entreprises et dans quels contextes la méthodologie peut entraîner un véritable gain;
2) Étudier les préférences des utilisateurs quant à l'exploration ontologique par la grille ou par l'onglet de requête graphique;
3) Étudier les préférences des usagers en fonction des tâches à accomplir entre le formulaire CRUD simplifiée et le formulaire CRUD à onglets;
4) Effectuer des études d'utilisabilité, de facilité d'apprentissage et de satisfaction générale d'OM.

7.6.5 Augmentation de la performance

1) Optimiser les algorithmes des bases de données en fonction d'OM;
2) Utiliser le chaînage arrière;
3) Reformuler une vue en une seule et même requête SPARQL.

7.6.6 Ajout de nouvelles fonctionnalités

1) Considérer les paradigmes de visualisation non utilisés (cartes, graphes, ligne du temps, réalité augmentée, etc.) pour l'évolution de l'application;
2) Permettre à OM de fonctionner avec des données orphelines d'ontologie, voire permettre de construire ou de modifier une ontologie en fonction des données observées;
3) Développer des mécanismes d'agrégation pour regrouper les individus dans les menus contextuels lorsque le graphe fourche dans la grille;

4) Trouver des moyens de faciliter la recherche dans les ontologies présentant une grande quantité de classes;
5) Ajouter les relations de subsomption;
6) Permettre l'insertion de calculs et de traitements sur les données à l'intérieur des ontologies de domaines à partir de l'interface.

7.6.7 Amélioration des fonctionnalités existantes

1) Améliorer la sélection des points d'entrée dans le(s) ontologie(s). (E.g., barre de recherche, séparation des classes par ontologies, etc. Voir Dadzie et Pietriga (2017));
2) Améliorer la coloration des en-têtes pour rendre plus évidents les liens entre les individus sur des patrons de plus de deux classes. En effet, lorsque trois classes sont étendues suivant le patron x -> y -> z, changer la valeur de la colonne objet de la classe x faisant le pont vers la classe y fera changer non seulement les valeurs de y mais aussi de z. Les prochaines versions de l'application devront prévoir un mécanisme visuel pour rendre évident ces liens entre les individus sur des patrons de plus de deux classes. On pourrait se servir, par exemple, de dégradés ou de couches de couleurs pour signifier la distance entre une classe et la classe pivot;
3) Séparer la fonctionnalité d'unicité des individus de la fonctionnalité de génération automatique d'étiquette en fonction des valeurs des propriétés des individus.

7.6.8 Personnalisation des applications

1) Permettre aux super-utilisateurs de choisir les fonctionnalités à utiliser dans chaque vue;
2) Permettre aux super-utilisateurs de modifier la position des composantes de l'interface;
3) Permettre aux super-utilisateurs de lier des propriétés des ontologies aux composantes;
4) Permettre une mécanique pour que l'usager puisse lui-même concevoir la hiérarchie d'un arbre de classes. Cette mécanique peut être mise en place par l'utilisation d'une composante générique d'arbre.

7.6.9 Utilisation des standards du Web sémantique

1) Tirer avantage de toutes les sémantiques RDF, RDFS et OWL;
2) Tendre vers l'utilisation de tous les types de données XSD;
3) Utiliser la forme standard RDF pour les énumérations dans les ontologies. Celle utilisée par OM est différente de celle préconisée par le standard OWL. L'implémentation a été faite de cette manière afin d'utiliser les mêmes fonctionnalités d'OM pour gérer les énumérations que pour gérer les données ordinaires. Cela dit, les versions subséquentes d'OM devraient tendre vers l'utilisation complète du standard OWL, afin de rendre la méthodologie plus universelle;
4) L'OV devrait tendre à devenir un des standards du WS et elle devrait être proposée à la communauté comme moyen d'uniformiser le développement d'API et de cadre de développement utilisant les données sémantiques.

7.6.10 Amélioration des mécanismes d'exploration

1) Développer des composantes permettant l'exploration sans douleur des ontologies où le nombre de branches possibles partant d'un même individu est très élevé;
2) Permettre à l'usager de choisir entre les différents codomaines d'une propriété lorsqu'il ajoute la classe à la grille. Cela permettra à l'usager de sélectionner n'importe quelle sous-classe de la hiérarchie.

ANNEXE I

CRV-HQ : INFORMATIONS SUPPLÉMENTAIRES SUR LE UI

L'interface client du CRV-HQ a été conçue itérativement. En construisant à partir des connaissances acquises pendant la réalisation de DATE, des fonctionnalités d'exploration et de tri ont tout d'abord été ajoutées, avant qu'une vaste gamme des possibilités offertes par SPARQL y soit implémentée de manière visuelle.

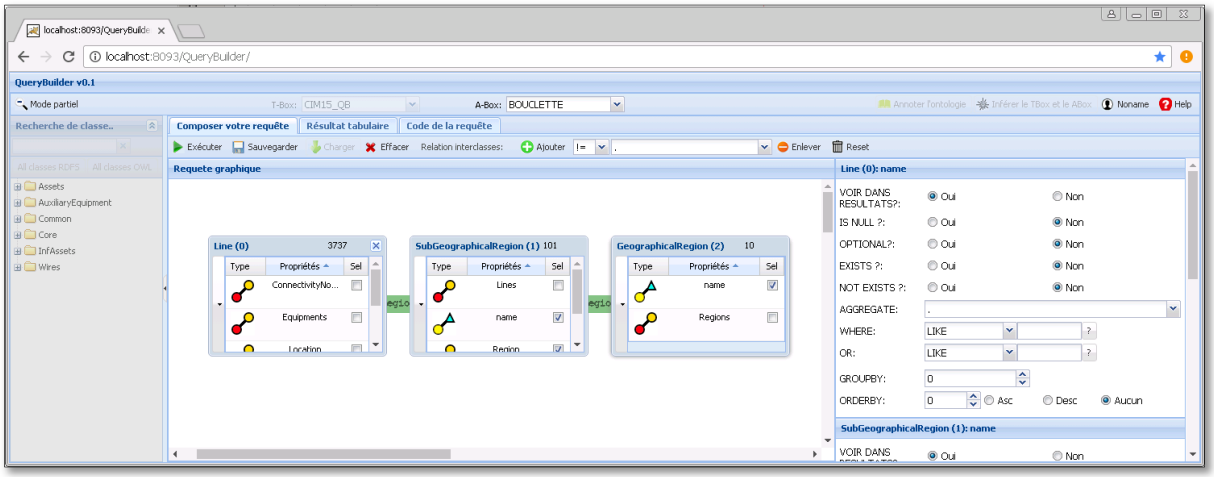

Figure-A I-1 Interface graphique du CRV-HQ

L'interface du CRV-HQ est séparée en quatre sections (Voir Figure-A I-1) : la barre des tâches, l'arbre des classes du modèle, l'espace de travail graphique et l'espace des formulaires de filtres.

La barre des tâches

À partir de la barre des tâches l'usager a accès à plusieurs fonctionnalités. Il peut y choisir l'ontologie (TBox) et le jeu de données (ABox) qu'il veut explorer. Des boutons lui permettent d'annoter ce TBox pour qu'il puisse être utilisé par le CRV-HQ ainsi que de faire appliquer les inférences sur l'agrégation du TBox et du ABox. L'usager pourra basculer entre le mode complet et le mode partiel, grâce au bouton qui se trouve à la gauche complètement de la barre

des tâches. Finalement, complètement à droite de la barre, est présenté le nom de l'usager, un bouton lui permettant de se déconnecter de l'application et un bouton d'aide permettant de télécharger le manuel de l'utilisateur.

L'arbre des classes

Lorsque l'usager choisit un TBox ou lorsqu'il bascule du mode partiel au mode complet, l'arbre des classes est chargé (ou rechargé). Si l'ontologie comporte des ballots de classes, l'arbre se génère grâce à un chargement paresseux qui rend cette opération plus rapide et plus pratique pour les recherches de l'usager.

L'arbre présente par défaut les classes du modèle séparées selon le ballot (*package*) qui les contient. Ce comportement a été choisi car il est le plus naturel pour naviguer dans l'ontologie des réseaux électriques, l'IEC/CIM. Des boutons ont été implémentés pour charger toutes les classes (RDFS ou OWL) sans séparation par ballot pour les ontologies n'en n'ayant pas.

Ce sont les étiquettes (*rdfs:label*) des classes qui sont utilisées comme nom autant pour les classes de l'ontologie que pour ses propriétés, qu'elles soient présentées dans l'arbre ou ailleurs. Tel que discuté précédemment, ces étiquettes sont nécessaires au bon fonctionnement de toutes les applications de cette recherche puisqu'elles sont lisibles par les humains (*human readable labels*).

L'arbre présente aussi un champ texte permettant d'effectuer des recherches sur les noms des classes présentes dans l'arbre.

Finalement, lorsque l'usager survol les classes ou les ballots de l'arbre avec le curseur de la souris, si l'ontologie contient un commentaire décrivant la classe celui-ci s'affiche dans une bulle contextuelle (*tool-tips*). La sémantique de commentaires du langage RDFS (*rdfs:comment*) permet d'ajouter ce type d'information sur les composantes d'une ontologie.

Onglet de travail graphique

L'espace de travail graphique se trouve dans le premier onglet de la partie centrale du UI comme présenté à la Figure-A I-1.

Il comprend une barre de tâche et un canevas graphique dans lequel les boites représentant les classes sont contenues. Voici comment fonctionnent ces dernières.

Sur sélection de la classe pivot de l'arbre des classes, l'arbre ainsi que le menu de sélection du TBox se grisent et deviennent inutilisables tant que l'espace de travail n'est pas nettoyé. L'arbre est grisé pour ne permettre qu'à une seule classe d'être la racine d'une requête, tandis que la sélection de TBox devient indisponible puisque le CRV-HQ ne fonctionne que sur un modèle à la fois. De plus, une boîte représentant la classe apparait dans l'espace graphique.

Comme pour l'arbre, si l'usager laisse sa souris en suspension au-dessus d'une boîte, une bulle contextuelle présente les commentaires (*rdfs:comment*) détaillant cette propriété.

Outre la liste des propriétés de la classe, on peut voir sur l'entête de la boîte, son nom, un numéro d'identification (entre parenthèses) et le nombre d'individus de cette classe contenus dans l'ABox. Changer d'ABox en cours de composition d'une requête entraîne la modification automatique de ce dénombrement.

Seule la première classe contient un bouton permettant de faire disparaître la boîte, ce qui entraîne la disparition de toutes les boîtes de la requête et la réactivation de l'arbre des classes et du menu des TBox.

À la gauche des boîtes représentant les classes, un bouton permet d'ouvrir un menu contextuel permettant de spécifier des clauses SPARQL sur la classe (voir Figure 4.8). Des boutons radios permettent :

1) De rendre cette classe optionnelle (OPTIONAL) (indisponible pour la classe pivot tel que vu dans la Figure 4.8);
2) D'afficher ou non une colonne correspondant à cette classe dans les résultats;
3) D'afficher le type de cette classe (voir explication plus bas);
4) De ne retrouver que les individus distincts (DISTINCT);
5) De n'afficher que les individus de la classe précédente où cette classe est instanciée;
6) De n'afficher que les individus de la classe précédente où cette classe n'est pas instanciée.

De plus, on peut ajouter les clauses GROUP BY et ORDER BY sur cette classe. Les champs de priorisation fonctionnent par l'utilisation de nombres entiers qui correspondent à l'ordre de priorité dans lequel les classes devraient être regroupées. L'ordre dans lesquels les priorisations sont faites inclut les classes et les propriétés dans la requête, indifféremment. Finalement, on peut agréger les classes grâce aux clauses COUNT, COUNT DISTINCT et GROUP CONCAT.

Si l'usager décide d'afficher le type d'une classe, la requête sera écrite de façon à retrouver le type le plus spécialisé de la hiérarchie de cet individu pour cette classe. Par exemple, si on sait que le disjoncteur numéro ABC123 est un équipement électrique, les disjoncteurs étant une sous-classe des équipements électriques, et que l'on demande tous les équipements électriques en voulant savoir leur type et leur numéro d'identification, le type de l'équipement dont le numéro est ABC123 sera "disjoncteur". Cette fonctionnalité peut s'avérer pratique dans les cas d'utilisation où il est important de déterminer le type précis d'un individu.

La barre des tâches de l'espace centrale permet d'exécuter, de sauvegarder, de télécharger ou d'effacer une requête du canevas graphique. Les sauvegardes prennent en considération la position des boîtes ainsi que leurs dimensions qui seront identiques lors du chargement.

À la droite de la barre des tâches de l'espace central, un utilitaire permet de définir des relations entre les classes. Cette fonctionnalité s'avère utile lorsque l'utilisateur compose une requête contenant deux fois la même classe. Il peut grâce à cette fonction spécifier si les individus des deux boîtes représentant la même classe sont identiques ou différents. Par exemple, si l'utilisateur compose une requête ayant le patron de classes : Équipement → Conducteur → Équipement, mais qu'il ne veut pas que soient affichés les cas où les équipements sont reliés à eux-mêmes, il pourra utiliser cette fonction pour spécifier que les individus des deux manifestations de la classe Équipement doivent être différents. Une petite interface a donc été ajoutée pour que l'usager puisse définir de telles relations interclasses et les gérer. La Figure-A I-2 illustre ce cas.

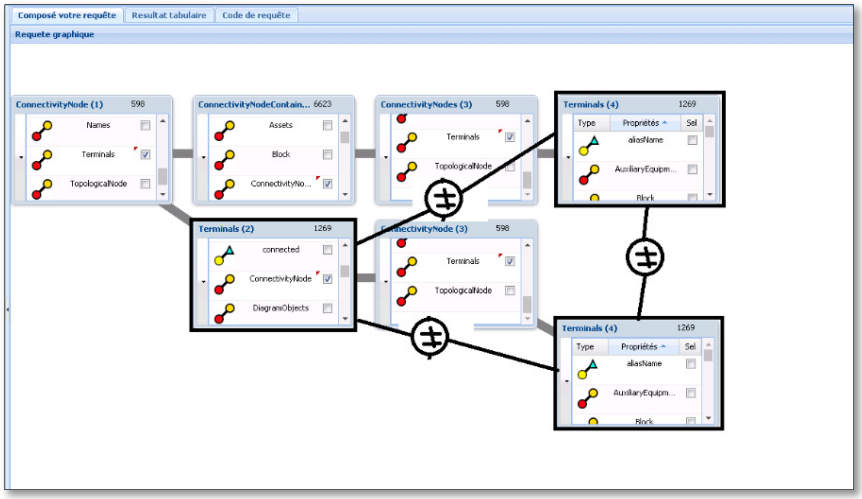

Figure-A I-2 Relations interclasses

Espace des formulaires de filtres

L'espace des formulaires de filtres se trouve à droite de l'espace de travail graphique, à l'intérieur du même onglet (voir Figure 4.6). Sa propre barre de défilement apparaîtra si l'utilisateur ajoute plus de filtres qu'il n'y a d'espace dans l'écran, ce qui lui permettra de parcourir ces filtres sans faire bouger le canevas graphique.

Les formulaires de filtres permettent d'ajouter des filtres sur les valeurs des relations de données des classes d'où ils sont issus. Ils sont constitués tout d'abord de cinq boutons radio qui permettent de :

1) Afficher ou non une colonne correspondante à cette relation dans les résultats;
2) Afficher les individus contenant explicitement une valeur NULL pour cette propriété;
3) Rendre cette propriété optionnelle (OPTIONAL);
4) N'afficher que les individus de cette classe dont cette propriété est instanciée;
5) N'afficher que les individus de cette classe dont cette propriété n'est pas instanciée.

Des menus déroulants permettent de choisir les options d'agrégation et les clauses WHERE, OR, GROUP BY et ORDER BY sur cette propriété. Le type cible du codomaine de cette propriété (*rdfs:range*) permet de n'afficher que les choix pertinents pour la propriété. Les types

reconnus sont le type texte (*xsd:string*), le type booléen (*xsd:boolean*), les types dates (*xsd:date* ou *xsd:dateTime*) et les types numériques (*xsd:float*, *xsd:integer*, *xsd:decimal* ou *xsd:double*).

Une propriété de type texte ou booléen permet les agrégations COUNT, COUNT DISTINCT et GROUP CONCAT, une date permet les mêmes en plus de MIN et MAX et un type numérique les permet tous en plus de AVG et SUM. La sélection d'une de ces options d'agrégation entraîne l'apparition d'un menu déroulant HAVING permettant de filtrer les résultats de cette agrégation.

De même, les clauses WHERE et OR ont des champs qui s'adaptent aux types des propriétés. Les types dates et numériques permettent les conditions !=, =, <, >, <=, >=, BETWEEN, NOT BETWEEN, IN et NOT IN. De plus, les champs de saisies des types dates sont des champs dates qui formulent les dates correctement et permettent l'utilisation d'un utilitaire de calendrier.

Les propriétés textes permettent les conditions LIKE, NOT LIKE, IN et NOT IN. Les propriétés booléennes quant à elles ne permettent que les conditions TRUE ou FALSE, et leur champ de saisie est indisponible.

Au bout des champs de saisie de ces clauses, un bouton étiqueté « ? » permet d'aider l'usager à choisir un filtre approprié. Ces boutons ouvrent des fenêtres contextuelles qui donnent des informations supplémentaires sur les modalités que prend cette propriété dans l'ABox. Les propriétés de type date ou numérique présentent les valeurs minimum et maximum trouvées dans l'ABox ainsi que la moyenne de toutes les valeurs. Les propriétés booléennes présentent le nombre de TRUE et le nombre de FALSE. Les propriétés textes quant à elles présentent les 100 premières modalités les plus retrouvées dans l'ABox ainsi que le nombre d'apparitions de celles-ci. En cas d'égalité, c'est l'ordre alphabétique qui détermine quelles modalités seront affichées. Même dans le cas où chaque modalité d'une propriété texte est unique, comme pour des numéros d'identification par exemple, l'affichage de celles-ci peut s'avérer utile pour guider l'usager dans la composition d'un filtre.

Finalement, les clauses GROUP BY et ORDER BY fonctionnent grâce à des champs prenant des nombres entiers qui servent à mettre en ordre de priorité les différentes propriétés faisant

l'objet d'un GROUP BY ou d'un ORDER BY dans la requête. La clause ORDER BY doit être spécifiée ascendante ou descendante pour être ajoutée à la requête.

Onglets des résultats et de la requête textuelle

Une fois la requête construite et exécutée, les résultats sont présentés dans l'onglet des résultats (Figure-A I-3). L'usager peut y ajuster le nombre de résultats qu'il désire voir par page et la page qu'il désire voir. Il peut aussi trier les résultats par colonnes, déplacer les colonnes ou faire disparaître des colonnes. Une fois la grille formatée, l'utilisateur peut exporter ces résultats au format CSV. La requête textuelle se trouve dans son propre onglet (Figure-A I-4).

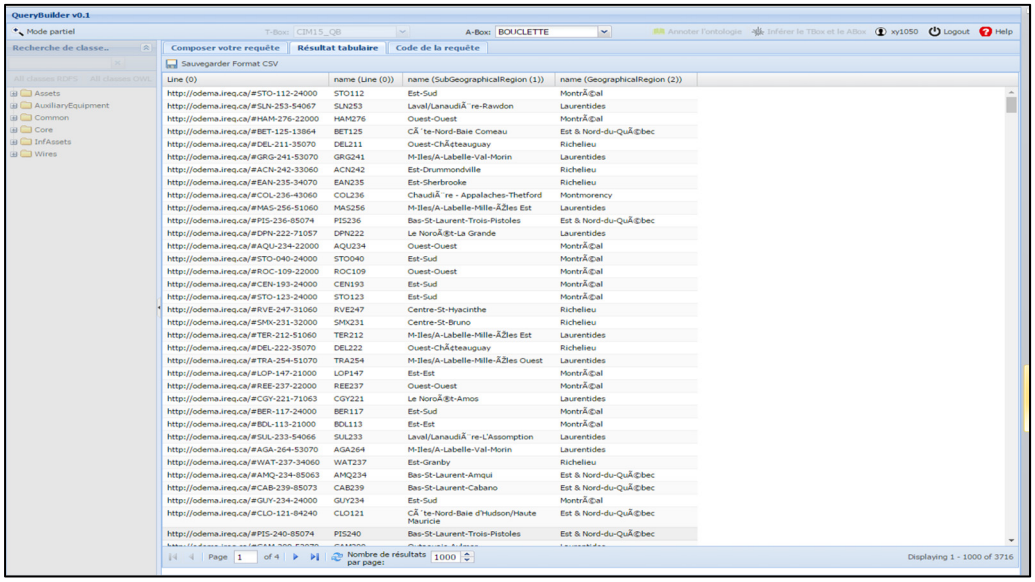

Figure-A I-3 Onglet des résultats

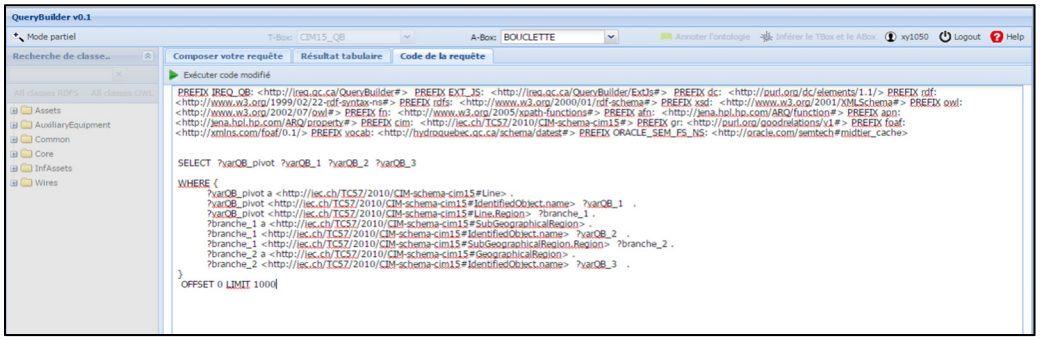

Figure-A I-4 Onglet de la requête textuelle

ANNEXE II

OM : INFORMATIONS SUPPLÉMENTAIRES SUR LE UI

L'interface d'OM contient quatre divisions principales tel qu'il a été présenté dans la Figure 5.1 : une barre de tâches, un arbre des vues et des classes, un panneau à onglet contenant trois onglets (onglet de la requête graphique, onglet des filtres et onglet des graphiques) et la grille sémantique. Voici des informations supplémentaires sur chacune de celles-ci.

Arbre des vues et des classes

Ce style de liste de classes en arbre est très semblable à celui du CRV-HQ, hormis que dans ce cas-ci, les classes ne sont pas regroupées par défaut dans des ballots. Dans le cas d'OM, la sélection de la classe pivot n'entraîne pas une désactivation de l'arbre. Au lieu de cela, le choix d'une autre classe de l'arbre entraîne le nettoyage de l'espace de travail et le remplacement de la requête actuelle par une nouvelle requête commençant par la dernière classe choisie dans l'arbre.

Les étiquettes (*rdfs:label*) des classes de cette liste sont présentées en ordre alphabétique. Lorsque disponibles, les commentaires *(rdfs:comment*) liés à ces classes sont présentés dans des bulles contextuelles (*tooltips*) lorsque le curseur s'attarde sur une classe.

L'arbre offre aussi quelques fonctionnalités pour la gestion des vues enregistrées. Ainsi, faire un clic droit sur une des vues enregistrées ouvre un menu contextuel permettant de supprimer ou de modifier cette vue enregistrée. Supprimer la vue de cette manière entraîne la suppression complète du graphe de cette sauvegarde de la BDT.

Modifier la vue entraîne tout d'abord le chargement de cette vue si elle n'est pas déjà chargée, puis l'ouverture de la fenêtre contextuelle de sauvegarde permettant de changer les informations entrées lors de la sauvegarde telles que le nom de la vue ou les permissions sur celle-ci. De plus, l'usager peut apporter des changements à sa vue, tels que l'ajout ou le retrait de classes ou de propriétés. La sauvegarde sera alors modifiée en conséquence.

La grille sémantique

Tel que mentionné, la grille sémantique présente sous forme d'un tableau en deux dimensions un graphe de données RDF. Des informations supplémentaires sur les colonnes, les outils sur les colonnes et sa barre des tâches seront maintenant présentés.

Les colonnes

Les colonnes de la grille correspondent à des propriétés de l'ontologie de domaine. L'OV peut être utilisée pour indiquer l'unité d'une propriété numérique. Cette unité est alors indiquée entre crochet à la suite de l'étiquette de la propriété dans l'entête de la colonne.

Les colonnes de types données contiennent des valeurs littérales. Si les valeurs de ces cellules contiennent des URL, un clic sur celles-ci provoque l'ouverture de l'adresse dans un nouvel onglet du navigateur. Les cellules contenant des chaînes de plus de 25 caractères sont, quant à elles, présentées dans des fenêtres flottantes afin de faciliter leur visualisation et permettre la sélection du texte pour les opérations de copier/coller.

Certaines colonnes présentent une icône en forme de fichier. Ces propriétés de type donnée ont été modélisées avec la propriété *om:file* provenant de l'OV. Les cellules de telles colonnes peuvent contenir un lien vers un fichier. Appuyer sur le contenu d'une telle cellule entraînera une réaction différente selon l'extension du fichier. Si le fichier est une image ou de type *Portable Document Format* (PDF), celui-ci sera présenté dans un nouvel onglet du navigateur. Pour toutes les autres extensions permises, le fichier sera téléchargé.

Les colonnes de types objets, quant à elles, lient les individus de deux classes. Pour améliorer l'utilisabilité de la grille, une colonne objet qui est étendue (i.e. dont la classe du codomaine est ajoutée à la grille) est automatiquement « clonée » et son clone est positionné au début des propriétés de la classe avec laquelle cette propriété fait le pont (voir Figure-A II-1). Cette décision a été prise parce que conceptuellement cette propriété est une propriété de la première classe. Cependant, comme c'est elle qui dirige les valeurs qui seront contenues dans les colonnes de la classe suivante, il est souvent plus aisé pour un utilisateur néophyte de saisir son utilité si elle est placée avec elles. C'est au super-utilisateur qui crée la vue de conserver ou de cacher l'une ou les deux colonnes grâce aux outils disponibles.

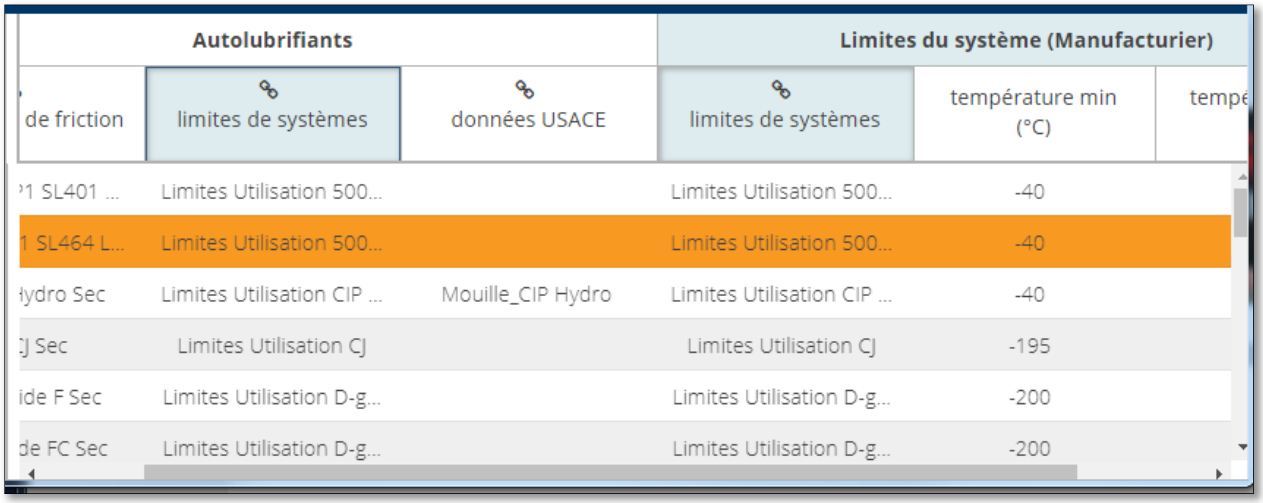

Figure-A II-1 Colonnes clonées

Il est possible de modifier l'ordre des classes dans la grille en glissant et en déposant les entêtes du niveau supérieur. De même, il est possible de changer l'ordre des propriétés à l'intérieur d'une même classe en glissant et en déposant les entêtes du niveau inférieur. Il n'est cependant pas permis de déplacer les propriétés d'une classe sous l'entête d'une autre classe.

Les outils sur les colonnes

Le menu contextuel d'actions pouvant être utilisées sur une colonne (voir Figure-A II-2) n'est pas identique selon le type de la propriété : il contient deux fonctionnalités supplémentaires lorsqu'il s'agit d'une colonne de type objet. Ce sont les fonctionnalités pour étendre et réduire une colonne. Ces fonctionnalités permettent d'ajouter ou de retirer les propriétés de la classe du codomaine de cette colonne dans la grille grâce au mécanisme qui a été présenté au CHAPITRE 5. Lors de l'ajout d'une classe toutes ses propriétés sont visibles et celles-ci sont placées complètement à droite des autres classes de la grille. Lorsqu'on enlève une classe de la grille, si celle-ci contenait elle-même des colonnes étendues (pointant vers d'autres classes affichées dans la grille), ces dernières sont également enlevées de la grille, à la manière d'une cascade. Ceci implique que toutes les classes qui succédaient à la classe qu'on vient de réduire

seront retirées de la grille. De mêmes, tous les filtres appliqués sur une colonne de la classe codomaine ou d'une des classes lui succédant seront détruits.

L'usager peut appliquer ces fonctionnalités à partir de deux autres endroits du UI : par les menus contextuels des cellules des colonnes objets ou en utilisant l'espace de requête graphique tel qu'il sera présenté plus loin.

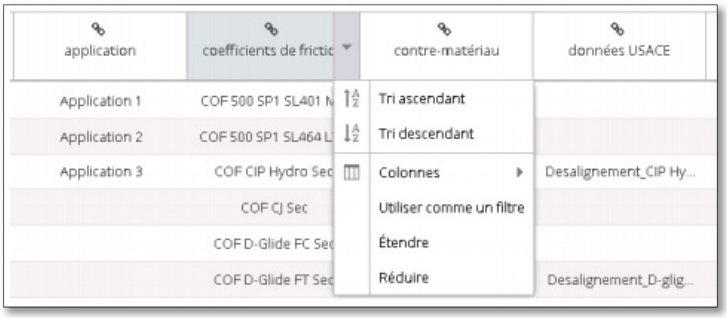

Figure-A II-2 Menu des outils sur les colonnes

Le menu des outils des colonnes permet aussi de trier la grille selon les valeurs d'une colonne de façon ascendante ou descendante. Plusieurs tris consécutifs peuvent être appliqués.

Il permet ensuite de cacher ou de montrer une colonne dans la grille. Les propriétés ainsi cachées ne sont pas enlevées de la base de données et seront toujours retournées avec le graphe de la classe, mais elles ne seront pas affichées. L'usager pourra réafficher la colonne en tout temps à partir du même menu, sans provoquer de requêtes sur la BDT.

Ce menu permet finalement d'appliquer un filtre sur la colonne afin de réduire l'espace de solution. Utiliser cette fonctionnalité fait apparaître un composant de filtre dans l'onglet des filtres du panneau central. Ce composant s'adapte selon le codomaine ou le type de la propriété (*rdfs:range* ou *rdf:type*) et sera donc différent s'il s'agit d'une date, d'un nombre, d'une chaîne de caractère ou d'un objet.

La barre des tâches de la grille sémantique

On retrouve dans la grille sémantique, différents outils pour agir sur celle-ci : les outils sur les lignes, les outils d'édition et de visualisation en formulaire et les autres outils sur la grille.

Les outils sur les lignes

Afin de pouvoir fabriquer n'importe quelle vue sur le graphe dans une grille en deux dimensions, il a été nécessaire d'ajouter ces outils sur les lignes (voir Figure-A II-3). Ces outils ne changent en rien le contenu de la BDT, ils ne sont utiles que pour l'affichage des données. Ceux-ci fonctionnent en association avec la colonne d'édition, tout à gauche de la grille sémantique. On sélectionne les lignes grâce à la colonne d'édition, puis on agit sur celles-ci grâce à ces outils.

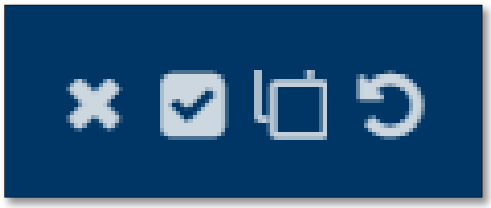

Figure-A II-3 Outils sur les lignes

Le premier outil permet de copier les lignes sélectionnées. L'étiquette de la nouvelle ligne contient un numéro entre parenthèses, correspondant au nombre de copies déjà faites de celle-ci. Copier une ligne peut être utile afin de, par exemple, visualiser simultanément deux individus différents d'une même classe liée à la classe pivot, comme le montre la Figure-A II-4.

Dans cet exemple, on voit que le matériau 500 SP1 SL464LT à deux jeux de coefficients de frictions, un lorsqu'il est sec et un lorsqu'il est mouillé. On peut alors copier la ligne (noter les cases de la colonne d'édition complètement à gauche qui sont cochées et le (2) qui apparaît à la suite de l'étiquette du matériau). Une fois la ligne copiée, l'usager peut choisir des valeurs différentes dans les cellules d'une colonne objet. Ici, il a sélectionné un jeu de coefficients de

friction différent dans chacune des deux lignes. L'usager peut ainsi construire un ensemble de solution présentant simultanément plusieurs branches du graphe.

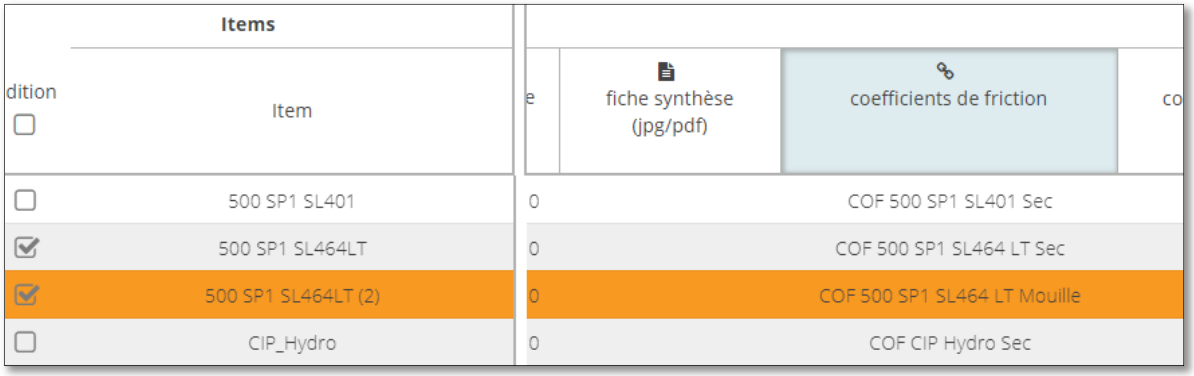

Figure-A II-4 Ligne copiée

D'ailleurs, lorsque l'usager choisit un individu parmi ceux présents dans une cellule d'une propriété objet, l'option 'Ouvrir tous les choix' apparaît. Utiliser cette option copie la ligne autant de fois qu'il y a de choix et affiche chacun des choix dans une ligne différente.

Le deuxième outil sur les lignes permet de cacher les lignes sélectionnées afin d'enlever des items non désirés de l'espace de solution. Le troisième outil fait l'inverse, il cache les lignes non-sélectionnées de façon à simplifier cette opération quand il y a plus de lignes à cacher qu'à montrer. Finalement, le dernier outil permet de réafficher les lignes qui ont été cachées dans la grille.

Les outils d'édition et de visualisation en formulaire

Les outils d'édition et de visualisation en formulaire (voir Figure-A II-5), font apparaître des fenêtres présentant une vue en formulaire des données.

Figure-A II-5 Outils d'édition

Les formulaires CRUD

Les deux premiers outils ayant beaucoup de choses en commun, ils seront présentés simultanément.

Le premier outil est appelé le CRUD à onglet (voir Figure 5.12). Il ouvre une fenêtre contenant autant d'onglets qu'il y a de classes dans la grille. Chacun de ces onglets peut alors être modifié individuellement. Les changements ainsi effectués peuvent être sauvegardés indépendamment des autres onglets.

Le deuxième outil est appelé le CRUD simplifié (voir Figure 5.13). Il ouvre une fenêtre contenant tous les formulaires (un pour chaque classe) un à la suite de l'autre dans un seul et même onglet. Chacun de ces formulaires est affublé des mêmes boutons et fonctionnalités que les onglets du CRUD à onglet. Une des principales différences entre les deux types de formulaires est que le CRUD simplifié n'a qu'un seul bouton pour sauvegarder les modifications et que celui-ci permet donc de sauvegarder tous les formulaires grâce à un seul clic. De son côté le CRUD à onglet contient un bouton de sauvegarde par onglet et chaque individu sera donc sauvegardé individuellement.

Les formulaires et leurs champs sont présentés, respectivement, dans l'ordre des classes et de leurs propriétés présentées dans la grille. Les usagers n'ont donc qu'à changer l'ordre des colonnes de la grille afin de modifier l'ordre des champs des formulaires. Les formulaires ne contiennent par défaut que les propriétés visibles de la grille au moment où les outils CRUD sont utilisés. En mode complet, un bouton (situé sur l'entête de la fenêtre) permet de rajouter les champs correspondant aux colonnes cachées dans les formulaires.

Le premier champ de chaque formulaire est appelé le champ d'étiquette (voir Figure-A II-6) et c'est un champ de type *combobox* présentant l'étiquette de l'individu sélectionné. Cela implique qu'un seul individu peut être affiché à la fois dans un formulaire. Le menu déroulant de ce champ montre tous les individus qu'il est possible de sélectionner pour ce formulaire. La sélection d'un individu dans ce champ fait afficher les valeurs des propriétés de cet individu dans le formulaire.

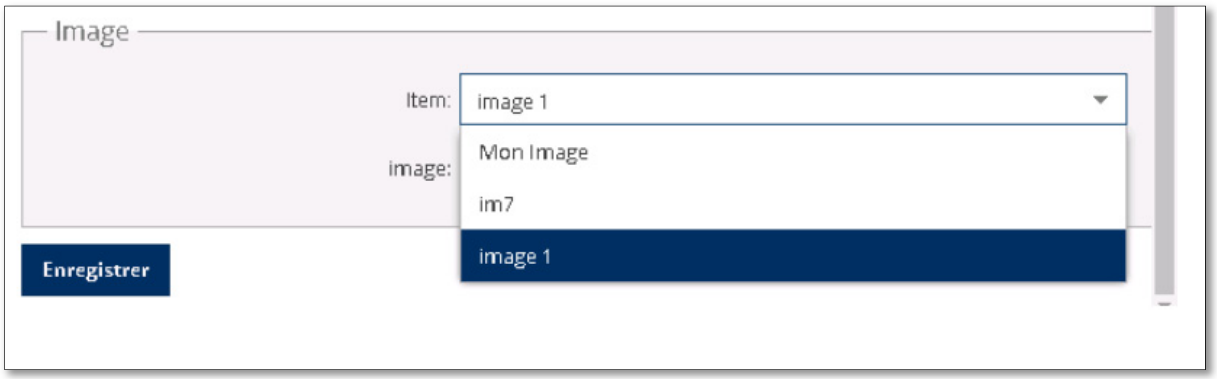

Figure-A II-6 Champ étiquette

Le champ d'étiquette d'une classe qui n'est pas la classe pivot est lié programmatiquement au champ objet de la propriété de la classe précédente faisant le pont entre ces deux classes. Tout changement à ce champ de la classe précédente est aussi appliqué aux choix disponibles dans le champ d'étiquette de la classe du codomaine. En effet, les choix du champ d'étiquette correspondent aux sélections effectuées dans l'autre champ. Les modifications s'appliquent automatiquement au fur et à mesure que l'usager rempli ces choix. Il va sans dire que les champs étiquettes liés à des propriétés fonctionnelles ne présentent en tout temps qu'un seul choix possible.

Les propriétés qui sont de type *owl:inverseOf* sont, quant à elles, potentiellement liées à des champs des classes précédentes. En effet, cette sémantique d'OWL implique qu'une relation reliant un sujet à un objet est l'inverse d'une autre relation reliant cet objet à ce sujet. Par exemple, la relation 'fils' étant l'inverse de la relation 'père', on pourra déduire que si Joe est le fils de Paul alors Paul est le père de Joe. Ainsi, lorsque la valeur d'un champ est modifiée et

que ce champ est l'inverse d'un autre champ, cet autre champ doit aussi être modifié en conséquence. Cela se fait aussi de manière automatique lorsque l'usager effectue des changements dans un formulaire.

Chaque formulaire est accompagné de trois boutons, correspondant à trois fonctionnalités différentes. Le premier permet d'ajouter un item. Cela signifie qu'il permet d'ajouter un nouvel individu à ce formulaire. S'il ne s'agit pas de la classe pivot, le champ objet de la classe précédente sera modifié en conséquence. Si le champ précédent est fonctionnel, un message d'alerte permettra à l'utilisateur de confirmer qu'il veut bel et bien remplacer le choix précédent par le nouveau.

Ce bouton permet à l'usager de choisir s'il veut créer un nouvel individu de toute pièce ou s'il veut ajouter un item déjà existant de la base de connaissance. Dans le premier cas, un nouvel URI lui sera attribué et il aura automatiquement une étiquette temporaire que l'usager sera appelé à modifier lors de la sauvegarde. Dans le deuxième cas, l'usager sera invité à choisir l'individu déjà existant à partir de la liste de tous les individus de cette classe contenus dans l'ABox.

Le deuxième bouton permet d'enlever un individu du formulaire. S'il ne s'agit pas de la classe pivot, le champ objet de la classe précédente sera modifié en conséquence. Ce bouton permet à l'usager de choisir s'il veut effacer l'individu uniquement de cette sauvegarde ou s'il veut effacer complètement l'individu de la BDT. Le premier choix n'élimine pas l'individu de la BDT, mais élimine uniquement le lien entre la classe précédente et cet individu. Le deuxième choix élimine l'item complètement de la BDT ainsi que toute référence à celui-ci.

Le troisième bouton permet de changer l'étiquette de cet individu. Le changement se répercute alors dans la grille et dans les autres champs des formulaires concernés. Il est possible à l'aide de l'OV d'annoter des classes afin que les noms de ses individus soient générés automatiquement en fonction des valeurs d'une ou de plusieurs de ses propriétés. L'OV permet d'indiquer si ce nom est alors obligatoire ou proposé. Si le nom est obligatoire, le bouton est alors désactivé et le changement de nom se fera automatiquement lors de la sauvegarde. Si le nom est proposé, l'utilisateur pourra activer le bouton et le nom généré automatiquement lui sera suggéré.

Unicité

L'OV permet aussi d'annoter des classes pour garantir l'unicité de ses individus en fonction de certaines de ses propriétés. Par exemple, on ne voudra pas dans une utilité électrique que deux individus différents de la classe « Pièce » référent à la même pièce du même groupe turbine-alternateur de la même centrale. L'usager tentant de créer un individu ayant une ou plusieurs propriétés annotées pour en garantir l'unicité se verra refuser l'opération s'il existe déjà dans la base de données un autre individu ayant la ou les mêmes valeurs à cette ou ces propriétés.

Si un individu modifié fait partie d'une classe où l'on doit garantir l'unicité de chaque individu, cette vérification est alors faite avant toute modification au modèle.

Changements concomitants

Afin d'éviter que des changements soient fait simultanément par deux usagers sur le même individu, des vérifications sont effectuées par le serveur. Celles-ci servent à déterminer si l'usager tente de modifier un triplet qui n'existe plus puisqu'il a été modifié par un autre utilisateur pendant l'envoie de la requête. Si tel est le cas, l'usager est averti de la situation et est appelé à rafraîchir son formulaire CRUD.

L'outil de visualisation du formulaire

Le troisième outil de cette série permet la visualisation en formulaire d'une ligne de la grille. Cet outil permet donc d'ouvrir l'équivalent d'un CRUD simplifié dans lequel l'usager ne peut rien modifier. De plus, les champs étiquettes ne sont plus des *combobox*; tous les individus possibles sont présentés un à la suite de l'autre en autant de formulaires, tel que présenté dans la Figure 5.15.

L'usager a alors la possibilité d'exporter cette forme de visualisation en format *Portable Network Graphics* (PNG) pour fin de référence ou pour être incorporée dans un autre document, tel qu'un rapport.

Les autres outils sur la grille

La dernière série d'outils permet de rafraîchir la grille, de sauvegarder une vue et d'exporter la grille au format CSV (voir Figure-A II-7).

Figure-A II-7 Autres outils

Le premier bouton permet donc de rafraîchir la grille. Les informations sur les classes sont redemandées à la BDT et la grille est régénérée complètement à partir de zéro. Cette fonctionnalité peut être utile, entre autres, lorsqu'on sait qu'un autre utilisateur est en train de faire des modifications sur la BDT et qu'on veut visualiser les nouveaux changements qu'il a effectués.

Le deuxième bouton permet de sauvegarder la vue. Ce bouton ouvre une fenêtre contextuelle de sauvegarde (voir Figure 5.18). Dans cette fenêtre, l'utilisateur doit entrer une étiquette à la sauvegarde et, optionnellement, un commentaire sur celle-ci pour la décrire (qui apparaîtra dans une bulle contextuelle dans la liste des classes). Il doit ensuite indiquer si la vue doit apparaître dans la liste des actions en mode simple. Permettre à une vue d'être partagée en mode simple signifie que tous les usagers pourront y accéder. Par opposition, ne pas partager la vue en mode simple signifie qu'uniquement les super-utilisateurs pourront y accéder.

Lorsque l'usager pèse sur le bouton 'Enregistrer', toutes les informations permettant de recharger la grille sont alors mises dans un graphe RDF. Ces informations sont : les colonnes étendues, les colonnes affichées, l'ordre des colonnes, les colonnes triées, la direction du tri et l'ordre des tris, les lignes cachées, les lignes copiées, les valeurs sélectionnées dans les cellules des colonnes objets, les filtres, leur type et les valeurs de leurs champs, le graphique créé dans l'onglet graphique et ses axes ainsi que l'identifiant de l'usager qui a créé la vue.

Finalement, le dernier outil de la grille sémantique permet de générer un fichier CSV à partir de la grille. Il ne présente que les colonnes présentement visibles dans la grille et uniquement les lignes qui ont été sélectionnées à partir de la colonne d'édition. Ce fichier est compatible avec *Excel* (Excel, 2020), ce qui est particulièrement utile dans un contexte d'entreprise.

Le panneau à onglets

Le panneau à onglets qui occupe la partie supérieure centrale de l'interface contient trois onglets : l'onglet de la requête graphique, l'onglet des filtres et l'onglet des graphiques.

Onglet de la requête graphique

Cet onglet présente les classes sous forme de boîtes comme c'était le cas dans le CRV-HQ. Dans OM, les boîtes ont deux colonnes d'actions. Les cases de la colonne V (pour visualiser) rendent visible la colonne de la grille liée à la propriété si elle est cochée et invisible si elle est décochée. De manière semblable, les cases de la colonne U (pour utiliser) permettent des actions sur les colonnes de la grille liées à la propriété. Ces actions diffèrent selon le type de la propriété. Ainsi, cocher la colonne U d'une propriété de type donnée fera apparaître un filtre lié à cette propriété dans l'onglet des filtres et la décocher fera disparaître ce filtre. Cocher la colonne U d'une propriété de type objet fera étendre la colonne de la grille et afficher la classe codomaine de celle-ci. La décocher fera l'action inverse, c'est-à-dire que la colonne sera réduite et que la classe codomaine de cette propriété n'apparaîtra plus dans la grille.

Onglet des filtres

Cet onglet présente les filtres que l'usager a choisis à partir de la grille ou de l'outil de requête graphique afin de réduire l'espace de solution. Il existe présentement cinq types de filtres correspondant aux types des codomaines des propriétés : les filtres numériques, les filtres date et date-temps, les filtres chaîne de caractères et les filtres objets.

Ceux-ci sont ré-ordonnables de gauche à droite et peuvent être enlevés grâce au bouton dans leur entête. Lorsque disponibles dans le TBox, les minimum et maximum de chaque filtre sont affichés dans leur titre.

Les filtres numériques permettent d'entrer une valeur maximale et une valeur minimale dans des champs numériques afin de ne ramener que les individus dont les valeurs de cette propriété se trouvent à l'intérieur de cette ou ces borne(s). Il est possible d'annoter les propriétés dans le TBox grâce à l'OV afin d'imposer un minimum et/ou un maximum à la propriété. De cette façon, la composante de filtre n'acceptera pas les valeurs au-delà ou en deçà de ceux-ci.

Les filtres dates et date-temps fonctionnent de la même manière que le filtre numérique, à l'exception que les champs d'entrée de données sont des champs date avec un utilitaire de calendrier pour aider la sélection. Comme c'est le cas pour les filtres numériques, l'OV peut être utilisée pour imposer un minimum, un maximum et un pas d'incrémentation.

Les filtres « chaînes de caractères » permettent d'entrer n'importe quel texte dans un champ texte. Les individus dont cette propriété aura des valeurs contenant la chaîne de caractères seront retournés. Un menu déroulant offre à l'utilisateur les possibilités de valeurs trouvées dans la base de données pour cette propriété, sans l'obliger à choisir une de celles-ci.

Un filtre objet force l'usager à sélectionner dans un menu déroulant l'étiquette d'un individu parmi la liste de tous les individus de cette classe trouvés dans la BDT. L'URI associé à cette propriété sera alors utilisé pour filtrer les individus de cette classe qui contiennent cet URI comme objet de la propriété filtrée.

Pour tous les types de filtres, si l'usager laisse un champ vide, celui-ci sera traité comme s'il n'y avait pas de filtre. Par exemple, un filtre numérique comporte deux champs : une valeur minimale et une valeur maximale. Si les deux sont vides, ce filtre n'est pas pris en compte. Si un des deux est vide, seule l'autre borne est prise en compte.

Onglet des graphiques

Cet onglet permet de créer un graphique sur les données de la grille. Le graphique à pointe de tarte, l'histogramme et le nuage de points de la librairie chart.js (Chart.js, 2020) ont été

implémentés. L'usager doit choisir tout d'abord le type de graphique, ensuite le ou les colonnes de la grille qu'il veut utiliser comme axes. Les choix possibles des axes sont présentés dans des menus déroulants et sont choisis en fonction du type et du codomaine des propriétés. Le graphique est ensuite généré à partir des lignes visibles de la grille.

La barre des tâches

La barre des tâches contient six actions possibles. Ces actions sont : changer de mode d'utilisation, se déconnecter, faire l'inférence, trouver une URI, télécharger un lot de données et demander de l'aide.

Le changement de mode d'utilisation permet de basculer entre le mode simple et le mode complet d'OM.

L'action de déconnexion permet à l'usager de fermer sa session et de retourner à la page de connexion.

L'action d'effectuer l'inférence permet d'exécuter les ensembles d'inférences RDFS et OWL sur le modèle et les données.

L'action pour trouver une URI, ouvre une fenêtre contextuelle dans laquelle l'usager peut taper l'étiquette d'un individu. Si cette étiquette est bien présente dans la base de données, l'URI de cet individu lui est alors retourné.

L'action de télécharger un lot de données, permet à un usager de télécharger un fichier OWL contenant un graphe de données se conformant au TBox de l'ontologie de référence dans l'ABox de la BDT. Cette fonction a été implémentée pour accommoder les chercheurs qui produisaient de grands lots de données depuis un banc d'essais électronique. Pour ne pas avoir à saisir ces données manuellement, un programme tiers a été utilisé pour transformer les données du banc d'essais en format OWL, permettant ainsi l'implémentation de cette fonctionnalité. Un service permet donc de charger ces lots de triplets dans la BDT. Ces lots doivent être enregistrés en format RDF/XML, dans des fichiers OWL réalisés grâce à l'utilitaire Cellfie (Cellfie, 2020) de Protégé (Protégé, 2020).

Finalement, le bouton d'aide entraîne le téléchargement du manuel de l'utilisateur.

ANNEXE III

ONTOLOGIE DE VISUALISATION

Cette annexe présente les propriétés de l'OV pour chacune de ses quatre classes de propriétés. Pour chaque propriété seront présentés une description de celle-ci ainsi que son codomaine qui permet de mieux saisir son fonctionnement.

Les propriétés d'annotation de classe

Ces propriétés servent à annoter les classes du modèle. Elles sont toutes des sous-propriétés de *om:propDeClasse*, afin de faciliter leur récupération.

Tableau-A III-1 Propriétés d'annotations de classe

URI	Description	Codomaine
om:autoGenererLabel	Cette propriété sert à indiquer que les étiquettes des individus de cette classe peuvent être autogénérées en se basant sur les valeurs des propriétés de celle-ci. S'utilise en conjonction avec la propriété d'annotation de propriété homonyme.	Nombre entier indiquant le nombre de propriétés entrant dans la génération de l'étiquette
om:forcerLabel	Cette propriété sert à indiquer que les étiquettes sont obligatoirement autogénérées. Son absence indique que l'étiquette auto-générée est proposée à l'utilisateur mais que le choix final lui appartient.	Booléen
om:obligerUniciteLabel	Cette propriété sert à indiquer que chaque individu de la classe doit présenter une combinaison unique de valeur aux propriétés entrant dans la conception de son étiquette.	Booléen

Puisque le premier cas d'utilisation (PPAL) ne nécessitait pas de distinction entre l'unicité de l'étiquette et l'unicité de l'individu, la propriété *om:obligerUniciteLabel* s'applique en conjonction avec *om:autoGenererLabel*. Des versions ultérieures de ce mécanisme devraient séparer ces deux concepts.

Les propriétés d'annotations de propriété

Ces propriétés servent à annoter les propriétés du modèle. Elles sont toutes des sous-propriétés de *om:propDeProp* afin de faciliter leur récupération.

Tableau-A III-2 Propriétés d'annotations de propriété

URI	Description	Codomaine
om:autoGenererLabel	S'utilisant avec les trois propriétés d'annotation de classe, cette propriété sert à indiquer que les étiquettes des individus de cette classe utilisent la valeur de cette propriété lors de leur génération.	Un nombre entier indiquant la position dans lequel la valeur de cette propriété entre dans la génération de l'étiquette
om:maxDeProp	Indique le maximum acceptable pour une valeur numérique de cette propriété.	Un nombre entier ou un nombre décimal, selon le type de la propriété
om:minDeProp	Indique le minimum acceptable pour une valeur numérique de cette propriété.	Un nombre entier ou un nombre décimal, selon le type de la propriété
om:pasDeProp	Indique un pas d'incrémentation pour une valeur numérique.	Un nombre entier ou un nombre décimal, selon le type de la propriété
om:unite	Indique l'unité des valeurs d'une propriété.	Chaîne de caractères
om:valeurAutoGeneree	Indique que les valeurs de cette propriété seront auto-générées et donc que l'usager ne peut modifier ces valeurs directement.	Booléen

Les propriétés d'annotations de sauvegarde

Ces propriétés servent à sauvegarder des vues et elles sont toutes des sous-propriétés de *om:propDeViz* afin de faciliter leur récupération. Elles sont séparées en :

1. Propriétés de méta-informations de sauvegarde;
2. Propriétés sur les valeurs des cellules objets de la grille;
3. Propriétés sur les lignes copiées ou cachées;
4. Propriétés sur les colonnes de la grille, leur ordre et si elles sont cachées;
5. Propriétés d'informations sur les tris sur les colonnes;
6. Propriétés d'informations sur les colonnes à étendre;
7. Propriétés d'informations sur les graphiques;
8. Propriétés d'informations sur les filtres.

Tableau-A III-3 Propriétés de méta-informations de sauvegarde

URI	Description	Codomaine
om:auteur	Indique le nom de l'usager ayant créé la sauvegarde	Chaîne de caractères
om:visualisation	Pointe vers l'URI des autres catégories d'information sur la sauvegarde	URI
om:commentaire	Contient un commentaire sur la sauvegarde.	Chaîne de caractères
om:simple	Indique qu'une sauvegarde doit être affichée en mode simple	Booléen
om:rolePerm	Indique quelles permissions sont données à quels rôles d'usager pour cette vue	URI
om:role	Rôle d'un om:rolePerm	Chaîne de caractères
om:permission	Permission d'un om:rolePerm	Chaîne de caractères
om:classeDeRef	URI de la classe pivot de la sauvegarde	URI

Tableau-A III-4 Propriétés sur les valeurs des cellules objets de la grille

URI	Description	Codomaine
om:comboBoxActivated	Pointe vers un nœud de sauvegarde pour les comboBox	URI
om:rowInstanceURI	URI de la ligne (i.e. l'URI de l'individu de la classe pivot) du om:comboBoxActivated	URI
om:columnPropURI	URI de la colonne (i.e. URI de la propriété représentée par la colonne) du om:comboBoxActivated	URI
om:comboPath	Suite d'URI séparés par des virgules des prédicats formant le chemin séparant la classe pivot et la classe à laquelle appartient ce om:comboBoxActivated	Chaîne de caractères
om:comboLabel	Étiquette à afficher dans le om:comboBoxActivated	Chaîne de caractères
om:comboValue	URI correspondant à l'étiquette à afficher dans le om:comboBoxActivated	URI

Tableau-A III-5 Propriétés sur les lignes copiées ou cachées

URI	Description	Codomaine
om:copyrow	Pointe vers un nœud de sauvegarde de lignes à copier	URI
om:deleterow	Pointe vers un nœud de sauvegarde de lignes à cacher	URI
om:instanceURI	URI de la ligne à cacher ou à copier de om:copyrow ou om:deleterow	URI

Tableau-A III-6 Propriétés sur les colonnes de la grille, leur ordre et si elles sont cachées

URI	Description	Codomaine
om:columnOrder	Pointe vers le nœud de sauvegarde d'ordre des colonnes	URI
om:propOuClasseURI	URI de la propriété ou de la classe	URI
om:orderPath	Suite d'URI séparés par des virgules des prédicats formant le chemin séparant la classe pivot de la classe à laquelle appartient cette colonne	Chaîne de caractères
om:orderNumber	Position de cette colonne dans la grille	Nombre entier
om:propOuClasse	Indique si la colonne est une classe (entête supérieure) ou une propriété (entête inférieure)	Chaîne de caractères ("prop" ou "classe")
om:shown	Indique si la colonne doit être montrée ou cachée	Booléen
om:colonneCopiee	Indique s'il s'agit d'une colonne copiée	Booléen

Tableau-A III-7 Propriétés d'informations sur les tris sur les colonnes

URI	Description	Codomaine
om:sortColumn	Pointe vers le nœud de sauvegarde de tri des colonnes	URI
om:propURI	URI de la propriété à trier	URI
om:sortDirection	Direction du tri	Chaîne de caractères ("ASC" ou "DESC")
om:sortColumnPath	Suite d'URI séparés par des virgules des prédicats formant le chemin séparant la classe pivot de la classe à laquelle appartient cette colonne	Chaîne de caractères
om:sortOrder	Ordre dans lequel doit être effectué ce tri par rapport aux autres tris	Nombre entier

Tableau-A III-8 Propriétés d'informations sur les colonnes à étendre

URI	Description	Codomaine
om:expandColumn	Pointe vers le nœud de sauvegarde des colonnes à étendre	URI
om:propURI	URI de la propriété de la colonne à étendre	URI
om:ecRange	Codomaine de la propriété de la colonne à étendre (i.e. la classe qui devra être retrouvée pour étendre la grille)	URI
om:pathExpand	Suite d'URI séparés par des virgules des prédicats formant le chemin séparant la classe pivot de la classe à laquelle appartient cette colonne	Chaîne de caractères
om:pathExpandRanges	Suite d'URI séparés par des virgules des codomaines formant le chemin séparant la classe pivot de la classe à laquelle appartient cette colonne	Chaîne de caractères
om:noOrdre	Ordre dans lequel cette colonne doit être étendue par rapport aux autres	Nombre entier

Tableau-A III-9 Propriétés d'informations sur les graphiques

URI	Description	Codomaine
om:graph	Pointe vers le nœud de sauvegarde du graphique	URI
om:graphChoice	Pointe vers les nœuds des choix effectués pendant la conception du graphique	URI
om:id	Id du champ du choix	Chaîne de caractères
om:value	Valeur choisie pour ce champ	Chaîne de caractères
om:order	Ordre dans lequel doivent être effectués les choix	Nombre entier

Tableau-A III-10 Propriétés d'informations sur les filtres

URI	Description	Codomaine
om:filtresTextes	Pointe vers le nœud de sauvegarde des filtres textes	URI
om:filtresNumeriques	Pointe vers le nœud de sauvegarde des filtres numériques	URI
om:filtresDate	Pointe vers le nœud de sauvegarde des filtres date	URI
om:filtresDateTime	Pointe vers le nœud de sauvegarde des filtres date-temps	URI
om:minInt	Valeur minimale choisie pour un filtre numérique de type entier	Nombre entier
om:maxInt	Valeur maximale choisie pour un filtre numérique de type entier	Nombre entier
om:minDec	Valeur minimale choisie pour un filtre numérique de type décimal	Nombre décimal
om:maxDec	Valeur maximale choisie pour un filtre numérique de type décimal	Nombre décimal
om:texte	Valeur choisie pour un filtre texte	Chaîne de caractères
om:from	Valeur minimale choisie pour un filtre date ou date-temps	Date ou date-temps
om:to	Valeur maximale choisie pour un filtre date ou date-temps	Date ou date-temps
om:pathFiltre	Suite d'URI séparés par des virgules des prédicats formant le chemin séparant la classe pivot de la classe à laquelle appartient cette colonne	Chaîne de caractères
om:pathFiltreRange	Suite d'URI séparés par des virgules des codomaines formant le chemin séparant la classe pivot de la classe à laquelle appartient cette colonne	Chaîne de caractères
om:stype	Indique si un filtre texte sert à une propriété de type donnée ou de type objet	Chaîne de caractères ("*owl:ObjectProperty*" ou "*owl:DatatypeProperty*")
om:dt	Indique le type de données du filtre (i.e. le codomaine de la propriété filtrée)	Chaîne de caractères
om:propFiltre	URI de la propriété filtrée	Chaîne de caractères
om:blank	Indique si le filtre doit accepter les valeurs nulles et, si oui, de quelle façon.	Chaîne de caractères ("*true*" pour un type superficiel, "*deep*" pour un type profond
om:ordreFiltre	Ordre dans lequel doivent être affichés les filtres	Nombre entier

Les propriétés d'annotations d'individus

Les propriétés d'annotations d'individus sont des propriétés qui s'appliquent sur tous les individus de la base de connaissances. Ce ne sont pas des propriétés d'annotations à proprement parler, puisqu'au sens de RDFS une propriété d'annotation ne participe pas à l'inférence. Elles sont donc différentes de toutes celles vues précédemment et sont en fait des propriétés de type données pour que l'inférence les associe à toutes les classes du modèle. Ainsi, en attribuant le domaine *owl:Thing* à ces propriétés, toutes les classes, grâce à l'ensemble de pré-inférences (voir Paradigmes de classification et d'héritage), héritent de ces propriétés.

Tableau-A III-11 Propriétés d'annotations d'individus

om:createurInstance	Indique qui a créé l'individu.	Chaîne de caractères
om:dateCreationInstance	Indique l'heure et la date de création	Date et Temps

Ces deux propriétés servent à assurer un suivi sur les modifications faites à la BDT. Tout individu ajouté à la BDT, que ce soit lors d'un ajout manuel ou lors de l'import en lot, reçoit automatiquement une valeur pour chacune de ces propriétés. Comme ces valeurs sont créées automatiquement et qu'elles ne doivent pas être modifiées par l'usager, on leur affuble la propriété *om:valeurAutoGeneree* (voir Tableau-A I-2). Ces propriétés s'utilisent en conjonction avec la permission modification restreinte. Une vue sauvegardée avec cette permission ne permet qu'au créateur d'un individu de pouvoir modifier son contenu.

RÉFÉRENCES BIBLIOGRAPHIQUES

Access. (2020, 06 24). Récupéré sur https://www.microsoft.com/fr-ca/microsoft-365/access

Ahmetaj, S., Calvanese, D., Ortiz, M., & Šimkus, M. (2014). Managing change in graph-structured data using description logics. *the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14)* (pp. 966–973). 966–973.

Allemang, D., & Hendler, J. (2008). *Semantic Web for the Working Ontologist.* Burlington: Denise E. M. Penrose.

Ambrus, O., Möller, K., & Handschuh, S. (2010). Konduit vqb: a visual query builder for sparql on the social semantic desktop. *Workshop on visual interfaces to the social and semantic web.*

Angular. (2020, 06 24). Récupéré sur https://angular.io/

Antoniou, G., Groth, P., Harmelen, F. v., & Hoekstra, R. (2012). *A Semantic Web Primer, Third Edition.* Cambridge, Massachusetts: The MIT Press.

Architecture trois tiers. (2020, June 24). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Architecture_trois_tiers

Athanasis, N., Christophides, V., & Kotzinos, D. (2004). Generating on the fly queries for the semantic web: The ics-forth graphical rql interface (grql). *International Semantic Web Conference* (pp. 486-501). 2004: Springer.

Bagosi, T., Calvanese, D., Hardi, J., Komla-Ebri, S., Lanti, D., Rezk, M., & Xiao, G. (2014). The ontop framework for ontology based data access. *Chinese Semantic Web and Web Science Conference2014* (pp. 67-77). Berlin: Springer.

Bawden, D., & Robinson, L. (2015). *Introduction to information science.* London: Facet Publishing.

Becker, C., & Bizer, C. (2008). DBpedia Mobile: A Location-Enabled Linked Data Browser. *Ldow*, 369.

Belleau, F., Nolin, M. A., Tourigny, N., Rigault, P., & Morissette, J. (2008). Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, 706-716.

Bermejo-Alonso, J., Sanz, R. R., & Hernández, C. (2010). Ontology-based engineering of autonomous systems. *Sixth International Conference on Autonomic and Autonomous Systems* (pp. 47-51). IEEE.

Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., & Sheets, D. (2006). Tabulator: Exploring and analyzing linked data on the semantic web. *3rd international semantic web user interaction workshop*, (p. 159).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific american, 284*(5), 34-43.

Berns, A., & Ghosh, S. (2009). Dissecting self-* properties. *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (pp. 10-19). IEEE.

Bhérer, L., Vouligny, L., Gaha, M., & Desrosiers, C. (2016). Toward an Adaptive Application Builder: Two Communication Systems for an Ontology-Based Adaptive Information System Framework. *International Journal On Advances in Intelligent Systems*, 9.1.

Bhérer, L., Vouligny, L., Gaha, M., Redouane, B., & Desrosiers, C. (2015). Ontology-based adaptive information system framework. *SEMAPRO 2015: The Ninth International Conference on Advances in Semantic Processing*, (pp. 110-115). Nice, France.

Card, M. (1999). *Readings in information visualization: using vision to think.* Morgan Kaufmann.

Cellfie. (2020, 06 24). Récupéré sur GitHub: https://github.com/protegeproject/cellfie-plugin

Chart.js. (2020, 06 24). Récupéré sur https://www.chartjs.org/

Chen, B., Wan, J., Shu, L., Li, P., M. M., & Yin, B. (2017). Smart factory of industry 4.0: Key technologies, application case, and challenges. *IEEE Access*, 6505-6519.

CIM. (2020, 06 24). Récupéré sur Wikipedia: https://en.wikipedia.org/wiki/Common_Information_Model_(electricity)

Control-M. (2020, 06 24). Récupéré sur bmc: https://fr.bmcsoftware.ca/it-solutions/control-m-capabilities.html

D2RQ. (2020, 06 24). Récupéré sur http://d2rq.org/

Dadzie, A. S., & Pietriga. (2017). Visualisation of linked data–reprise. *Semantic Web*, 1-21.

Dadzie, A. S., & Rowe, M. (2011). Approaches to visualising linked data: A survey. *Semantic Web*, 89-124.

Dadzie, A. S., Rowe, M., & Petrelli, D. (2011). Hide the stack: toward usable linked data. *Extended Semantic Web Conference* (pp. 93-107). Berlin: Springer.

Définition de système d'information. (2020, June 24). Récupéré sur Les définitions: le dico des définitions: https://lesdefinitions.fr/systeme-dinformation

Disco. (2020, 06 24). Récupéré sur http://wifo5-03.informatik.uni-mannheim.de/bizer/ng4j/disco/

Dobson, S., Sterritt, R., Nixon, P., & Hinchey, M. (2010). Fulfilling the vision of autonomic computing. *Computer, 43*(1), 35-41.

Docker. (2020, 06 24). Récupéré sur https://www.docker.com/

Données ouvertes. (2020, June 24). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Donn%C3%A9es_ouvertes

EJB. (2020, 06 24). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Enterprise_JavaBeans

Ermilov, I., Lehmann, J., Martin, M., & Auer, S. (2016). LODStats: The data web census dataset. *International Semantic Web Conference*, 38-46.

Excel. (2020, 06 24). Récupéré sur Microsoft: https://www.microsoft.com/en-us/microsoft-365/excel

Floridi, L. (2002). On defining library and information science as applied philosophy of information. *Social epistemology, 16*(1), 37-49.

Fresnel. (2020, 06 24). Récupéré sur https://www.w3.org/2005/04/fresnel-info/

Gaha, Zinflou, A., Bouffard, A., Vouligny, L., Viau, M., Langheit, C., & Martin, E. (2015). Temporal RDF System for Power Utilities. *9th International Conference on Advances in Semantic Processing - SEMAPRO 2015*, (pp. 32-37). Nice, France.

Gaha, Zinflou, A., Langheit, C., Bouffard, A., Viau, M., & Vouligny, L. (2013). An ontology-based rea-soning approach for electric power utilities. *Web Reasoning and Rule Systems - 7th International Conference*, (pp. 95–108). Mannheim, Germany.

Gherbi, T., Meslati, D., & Borne, I. (2009). MDE between Promises and Challenges. *2009 11th International Conference on Computer Modelling and Simulation* (pp. 152-155). IEEE.

Giese, M., Soylu, A., Vega-Gorgojo, G., Waaler, A., Haase, P., Jiménez-Ruiz, E., & Rosati, R. (2015). Optique: Zooming in on big data. *Computer*, 60-67.

GoodRelations. (2020, 06 24). Récupéré sur http://www.heppnetz.de/projects/goodrelations/

Grafkin, P., Mironov, M., Fellmann, M., Lantow, B., Sandkuhl, K., & Smirnov, A. V. (2016). SPARQL Query Builders: Overview and Comparison. *BIR Workshops*, 255-274.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies, 43*(5-6), 907-928.

Guimaraes, V. T., Freitas, C. M., Sadre, R., Tarouco, L. M., & Granville, L. Z. (2015). A survey on information visualization for network and service management. *IEEE Communications Surveys & Tutorials*, 285-323.

Haag, F., Lohmann, S., & Ertl, T. (2014). SparqlFilterFlow: SPARQL query composition for everyone. *European Semantic Web Conference* (pp. 362-367). Cham: Springer.

Haag, F., Lohmann, S., Bold, S., & Ertl, T. (2014). Visual SPARQL querying based on extended filter/flow graphs. *The 2014 International Working Conference on Advanced Visual Interfaces*, (pp. 305-312).

Hachey, G., & Gasevic, D. (2011). *Semantic web user interfaces: A systematic mapping study.* Athabasca University.

Heim, P., & Ziegler, J. (2009). Faceted visual exploration of semantic data. *Workshop on Human-Computer Interaction and Visualization* (pp. Workshop on Human-Computer Interaction and Visualization). Berlin: Springer.

Heim, P., Ertl, T., & Ziegler, J. (2010). Facet graphs: Complex semantic querying made easy. *Extended Semantic Web Conference* (pp. 288-302). Berlin: Springer.

Héon, M., Nkambou, R., & Gaha, M. (2016). OntoCASE4G-OWL: Towards a modeling software tool for G-OWL a visual syntax for RDF/RDFS/OWL2. *15th International Semantic Web Conference.* Kobe .

Heylighen, F. (2020, June 24). *Information system.* Récupéré sur Web Dictionary of Cybernetics and Systems: http://pespmc1.vub.ac.be/ASC/INFORM_SYSTE.html

Hildebrand, M., Van Ossenbruggen, J., & Hardman, L. (2006). /facet: A browser for heterogeneous semantic web repositories. *International Semantic Web Conference* (pp. 272-285). Berlin: Springer.

Hirst, P. H. (1974). *KNOWLEDGE AND THE CURRICULUM: A COLLECTION OF PHILOSOPHICAL PAPERS.* London: Routledge & Kegan Paul.

Horn, P. (2001). Autonomic computing: IBM’s perspective on the state of information technology.

Huynh, D. F., & Karger, D. (2009). Parallax and companion: Set-based browsing for the data web. *WWW Conference. ACM.*

Hydro-Québec. (2020, 06 24). Récupéré sur http://www.hydroquebec.com/residentiel/

Hyvönen, E. (2004). Finnish museums on the semantic web: the user's perspective on MuseumFinland. *Archives & Museums Informatics*.

Information system. (2020, 06 24). Récupéré sur BusinessDictionary: http://www.businessdictionary.com/definition/information-system.html

inova8. (2020, 06 24). Récupéré sur http://inova8.com/bg_inova8.com/

IREQ. (2020, 06 24). Récupéré sur http://www.hydroquebec.com/innovation/fr/institut-recherche.html

Jara, A. J., Olivieri, A. C., Bocchi, Y., Jung, M., Kastner, W., & Skarmeta, A. F. (2014). Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence. *International Journal of Web and Grid Services*, 244-272.

Jarrar, M., & Dikaiakos, M. D. (2008). *MashQL: A Query-By-Diagram Language for Data Mashups.* University of Cyprus.

Jarrar, M., & Dikaiakos, M. D. (2008). MashQL: a query-by-diagram topping SPARQL. *The 2nd international workshop on Ontologies and information systems for the semantic web*, (pp. 89-96).

Jarrar, M., & Dikaiakos, M. D. (2009). A data mashup language for the data web.

Jarrar, M., & Dikaiakos, M. D. (2011). A query formulation language for the data web. *IEEE Transactions on Knowledge and Data Engineering*, 783-798.

Java. (2020, 06 24). Récupéré sur https://www.java.com/fr/about/

Javascript. (2020, 01 27). Récupéré sur Mozilla: https://developer.mozilla.org/fr/docs/Web/JavaScript

Jena. (2020, 06 24). Récupéré sur Apache: https://jena.apache.org/

Jenkins. (2020, 06 24). Récupéré sur https://www.jenkins.io/

JSON-LD. (2020, 06 24). Récupéré sur https://json-ld.org/

Karger, D. (2006). The pathetic fallacy of RDF.

Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., & Giannopoulou, E. (2007). Ontology visualization methods—a survey. *ACM Computing Surveys (CSUR)*, 10-es.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer, 36*(1), 41-50.

Kobilarov, G., & Dickinson, I. (2008). Humboldt: Exploring linked data. *Context*.

kubernetes. (2020, 06 24). Récupéré sur https://kubernetes.io/fr/

Learn about DBPedia. (2020, June 24). Récupéré sur DBPedia: https://wiki.dbpedia.org/about

Lee, J., Bagheri, B., & Kao, H. A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*, 18-23.

LENA. (2020, 06 24). Récupéré sur https://west.uni-koblenz.de/research/software/lena

Linked data. (2020, June 24). Récupéré sur W3C: https://www.w3.org/standards/semanticweb/data

Liu, S., Cui, W., Wu, Y., & Liu, M. (2014). A survey on information visualization: recent advances and challenges. *The Visual Computer*, 1373-1393.

Mangold, C. (2007). A survey and classification of semantic search approaches. *International Journal of Metadata, Semantics and Ontologies*, 23-34.

Marbles. (2020, 06 24). Récupéré sur http://mes.github.io/marbles/

Marie, N., & Gandon, F. (2014). Survey of linked data based exploration systems. *WIMMICS - Web-Instrumented Man-Machine Interactions, Communities and Semantics.*

McGinnes, S., & Kapros, E. (2015). Conceptual independence: A design principle for the construction of adaptive information systems. *Information Systems*(47), 33-50.

MDA. (2020, 06 24). Récupéré sur OMG: https://www.omg.org/mda/

Medhe, S., & Phalke, D. A. (2012). RDF data retrieval in structured format using aggregate function and keyword search in MashQL.

Microsoft Access. (2020, 6 24). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Microsoft_Access

Noadster. (2020, 06 24). Récupéré sur https://homepages.cwi.nl/~media/demo/noadster/

OMG. (2020, June 24). Récupéré sur https://www.omg.org/

OpenShift. (2020, 06 24). Récupéré sur RedHat: https://www.openshift.com/

OpenUI5. (2020, 06 24). Récupéré sur https://openui5.org/

Oracle. (2020, 06 24). Récupéré sur https://docs.oracle.com/database/121/index.html

OWL. (2020, June 24). Récupéré sur W3C: https://www.w3.org/2001/sw/wiki/OWL

Pan, J., Staab, S., Aßmann, U., & Ebert, J. Z. (2013). *Ontology-Driven Software Development.* Springer-Verlag Berlin Heidelberg.

Paquette, G. (2010). *Visual Knowledge Modeling for Semantic Web Technologies: Models and Ontologies.* Information Science Reference.

Peinl, R. (2016). Semantic web: State of the art and adoption in corporations. *KI-Künstliche Intelligenz*, 131-138.

PL/SQL. (2020, 06 24). Récupéré sur https://www.oracle.com/ca-en/database/technologies/appdev/plsql.html

PoLA. (2020, 06 24). Récupéré sur Wikipedia: https://en.wikipedia.org/wiki/Principle_of_least_astonishment

Popov, I. O., Schraefel, M. C., Hall, W., & Shadbolt, N. (2011). Connecting the dots: a multi-pivot approach to data exploration. *International semantic web conference* (pp. 553-568). Berlin: Springer.

Protégé. (2020, 06 24). Récupéré sur https://protege.stanford.edu/

RDF. (2020, June 24). Récupéré sur W3C: https://www.w3.org/2001/sw/wiki/RDF

RDF Gravity. (2020, 06 24). Récupéré sur http://tapor.ca/tools/363

RDF Platform. (2020, 6 24). Récupéré sur https://www.ebi.ac.uk/rdf/

RDFS. (2020, June 24). Récupéré sur W3C: https://www.w3.org/2001/sw/wiki/RDFS

RelFinder. (2020, 06 24). Récupéré sur http://www.visualdataweb.org/relfinder.php

REST. (2020, 06 24). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Representational_state_transfer

Rich Internet application. (2020, 06 24). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Rich_Internet_application

Rutledge, L., Van Ossenbruggen, J., & Hardman, L. (2004). Making RDF presentable: integrated global and local semantic Web browsing. *Proceedings of the 14th international conference on World Wide Web,.* Chiba.

Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, (pp. 1-42).

Saracevic, T. (2009). Information science. Dans F. Heylighen, J. Bates, & M. N. Maack, *Encyclopedia of library and information sciences.* (pp. 2570-2586). New York: Taylor & Francis.

Schmidt, D. C. (2006). Model-driven engineering. *IEEE Computer*.

Schweiger, D. (2014). SPARQLGraph: Graphical query builder for biological federated SPARQL queries. *PhD Thesis*. Private University for Health Sciences, Medical Informatics and Technology.

Schweiger, D., Trajanoski, Z., & Pabinger, S. (2014). SPARQLGraph: a web-based platform for graphically querying biological Semantic Web databases. *BMC bioinformatics*, 1-5.

SEAMS. (2020, Juin 24). Récupéré sur https://conf.researchr.org/home/seams-2020

Semantic Web. (2020, June 24). Récupéré sur W3C: https://www.w3.org/standards/semanticweb/

Sencha Ext JS. (2020, 06 24). Récupéré sur https://www.sencha.com/products/extjs/

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. *1996 IEEE symposium on visual languages* (pp. 336-343). IEEE.

Shneiderman, B., & Plaisant, C. (2010). *Designing the user interface: strategies for effective human-computer interaction.* Pearson Education India.

SIMILE. (2020, 06 24). Récupéré sur http://simile.mit.edu/

Sowa, J. F. (2011). Serious semantics, serious ontologies. *Semantic Technology Conference (SEMTEC 2011).* San Francisco.

Soylu, A., & Giese, M. (2016). Qualifying ontology-based visual query formulation. Dans *Flexible Query Answering Systems 2015* (pp. 243-255). Cham: Springer.

Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., & Horrocks, I. (2013). OptiqueVQS: towards an ontology-based visual query system for big data. *Fifth International Conference on Management of Emergent Digital EcoSystems*, (pp. 119-126).

Soylu, A., Giese, M., Jimenez-Ruiz, E., Vega-Gorgojo, G., & Horrocks, I. (2016). Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. *Universal Access in the Information Society*, 129-152.

Soylu, A., Kharlamov, E., Z. D., Jimenez-Ruiz, E., Giese, M., & Horrocks, I. (2015). OptiqueVQS: Ontology-Based Visual Querying. *VOILA@ ISWC*, (p. 91).

Soylu, A., Kharlamov, E., Zheleznyakov, D., Jimenez-Ruiz, E., Giese, M., & Horrocks, I. (2014). OptiqueVQS: Visual query formulation for OBDA. *CEUR Workshop Proceedings.*

Soylu, A., Skjæveland, M. G., Giese, M., Horrocks, I., J.-R. E., Kharlamov, E., & Zheleznyakov, D. (2013). A preliminary approach on ontology-based visual query formulation for big data. *Research Conference on Metadata and Semantic Research* (pp. 201-212). Cham: Springer.

SPARQL. (2020, June 24). Récupéré sur W3C: https://www.w3.org/2001/sw/wiki/SPARQL

Spatial and Graph. (2020, 06 24). Récupéré sur Oracle: https://docs.oracle.com/database/121/RDFRM/toc.htm

SPIN. (2020, 06 24). Récupéré sur https://spinrdf.org/

SPIN Technology Stack. (2020, 06 24). Récupéré sur https://spinrdf.org/spinstack.html

Spring. (2020, 06 24). Récupéré sur https://spring.io/projects/spring-framework

Sterritt, R., & Hinchey, M. (2010). SPAACE IV: Self-properties for an autonomous & autonomic computing environment–Part IV A Newish Hope. *2010 Seventh IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems* (pp. 119-125). IEEE.

SWP. (2020, 06 24). Récupéré sur https://uispin.org/

The Linked Open Data Cloud. (2020, June 24). Récupéré sur https://lod-cloud.net/

Thuluva, A. S., Anicic, D., & Rudolph, S. (2017). Semantic Web of Things for Industry 4.0. *RuleML+ RR (Supplement)*.

Toad (logiciel). (2020, 6 24). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Toad_(logiciel)

Toad Data Point. (2020, 6 24). Récupéré sur Toad World: https://www.toadworld.com/products/toad-data-point

Toad for Oracle. (2020, 06 24). Récupéré sur Toad World: https://www.toadworld.com/products/toad-for-oracle

Toad World. (2020, 06 24). Récupéré sur https://www.toadworld.com/

Tomcat. (2020, 06 24). Récupéré sur Apache: https://tomcat.apache.org/download-70.cgi

TomEE. (2020, 06 24). Récupéré sur Apache: http://tomee.apache.org/apache-tomee.html

TopBraid. (2020, 06 24). Récupéré sur TopQuadrant: https://www.topquadrant.com/products/topbraid-enterprise-data-governance/

TopQuadrant. (2020, 6 24). Récupéré sur https://www.topquadrant.com/

Tory, M., & Moller, T. (2004). Rethinking visualization: A high-level taxonomy. *IEEE Symposium on information visualization*, 151-158.

Vaidyaa, S., Ambadb, P., & Bhoslec, S. (2018). Industry 4.0–a glimpse. *Procedia Manufacturing*, 9789.

Vega-Gorgojo, G., Slaughter, L., Giese, M., Heggestøyl, S., Soylu, A., & Waaler, A. (2016). Visual query interfaces for semantic datasets: An evaluation study. *Journal of Web Semantics*, 81-96.

Veres, C., Johansen, K., & Opdahl, A. L. (2010). Browsing and visualizing semantically enriched information resources. *2010 International Conference on Complex, Intelligent and Software Intensive Systems* (pp. 968-973). IEEE.

Vogel-Heuser, B., & Hess, D. (2016, April). Industry 4.0-Prerequisites and Visions (Guest Editorial). *IEEE Transactions on Automation Science and Engineering, 13*(2), 411-413.

Vouligny, L., & Robert, J. M. (2005). Online help system design based on the situated action theory. *Proceedings of the 2005 Latin American conference on Human-computer interaction*, (pp. 64-75).

Vouligny, L., Hudon, C., & Nguyen, D. N. (2009). Design of MIDA, a Web-Based Diagnostic Application for Hydroelectric Generators. *2009 Fourth International Multi-Conference on Computing in the Global Information Technology* (pp. 98-103). IEEE.

W3C. (2020, 06 24). Récupéré sur https://www.w3.org/

Ward, M. O., Grinstein, G., & Keim, D. (2010). *Interactive data visualization: foundations, techniques, and applications.* CRC Press.

Wikipedia. (2020, June 24). Récupéré sur https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal

Wilson, M., Russell, A., Schraefel, M. C., & Smith, D. A. (2006). Mspace mobile: A ui gestalt to support on-the-go info-interaction. *CHI'06 extended abstracts on Human factors in computing systems*, (pp. 247-250).

Wu, Z., Xu, Y., Yang, Y., Zhang, C., Zhu, X., & Ji, Y. (2017). Towards a semantic web of things: a hybrid semantic annotation, extraction, and reasoning framework for cyber-physical system. *Sensors*, 403.

Zinflou, A., Gaha, M., Bouffard, A., Vouligny, L., Langheit, C., & Viau, M. (2013, September). Application of an ontology-based and rule-based model in electric power utilities. *2013 IEEE Seventh International Conference on Semantic Computing* (pp. 405-411). Irvine, CA, USA: IEEE.

Zviedris, M., Romane, A., Barzdins, G., & Cerans, K. (2014). Ontology-Based Information System. *Semantic Technology*, 33-47.

Zwass, V. (2020, 06 24). *Information system.* Récupéré sur Encyclopedia Britannica: https://www.britannica.com/topic/information-system