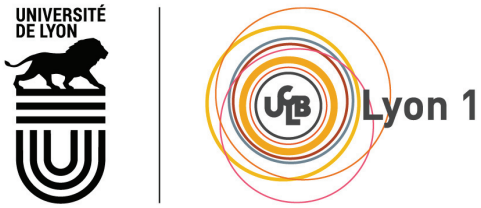
Formalisation automatique et sémantique de règles métiers

Cheikh Hito Kacfah Emani



N° d'ordre NNT : 2016LYSE1301

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
l'Université Claude Bernard Lyon 1

École Doctorale IED512
Informatique et Mathématiques

Spécialité de doctorat : Informatique
Discipline : Informatique

Soutenue publiquement le 01/12/2016, par :
Cheikh Hito KACFAH EMANI

Formalisation automatique et sémantique de règles métiers

Devant le jury composé de :

Nazarenko Adeline, Professeur des Universités, Paris 13	Rapporteure
Bellatreche Ladjel, Professeur des Universités, ENSMA	Rapporteur
Koukam Abderrafiaa, Professeur des Universités, UTBM	Examinateur
Le Thanh Nhan, Professeur des Universités, UNSA	Examinateur
Fiès Bruno, Ingénieur R&D, CSTB Sophia Antipolis	Examinateur
Ghodous Parisa, Professeur des Universités, Lyon 1	Directrice de thèse
Ferreira Da Silva Catarina, Maître de Conférences, Lyon 1	Co-directrice de thèse

À mes grands-mères Djambou et Emgoue.
Elles ne comprendraient sans doute pas le contenu de ce manuscrit, mais qu'elles le reçoivent comme des remerciements pour leur sagesse et leurs valeurs transmises à tant de générations.

Résumé

Cette thèse porte sur la transformation automatique et sémantique de règles métiers en des règles formelles. Ces règles métiers sont originellement rédigées sous la forme de textes en langage naturel, de tableaux et d'images. L'objectif est de mettre à la disposition des experts métiers, un ensemble de services leur permettant d'élaborer des corpus de règles métiers formelles. Le domaine de la Construction est le champ d'application de ces travaux. Disposer d'une version formelle et exécutable de ces règles métiers servira à effectuer des contrôles de conformité automatique sur les maquettes numériques des projets de construction en cours de conception.
Pour cela, nous avons mis à disposition des experts métiers les deux principales contributions de cette thèse. La première est la mise sur pied d'un langage naturel contrôlé, dénommé RAINS. Il permet aux experts métiers de réécrire les règles métiers sous la forme de règles formelles. Les règles RAINS se composent de termes du vocabulaire métier et de mots réservés tels que les fonctions de comparaisons, les marques de négation et de quantification universelle et les littéraux. Chaque règle RAINS a une sémantique formelle unique qui s'appuie sur les standards du web sémantique. La seconde contribution majeure est un service de formalisation des règles métiers. Ce service implémente une approche de formalisation proposée dans le cadre de cette thèse et dénommée FORSA. Ce service propose des versions RAINS des règles métiers en langage naturel qui lui sont soumises. FORSA fait appel à des outils du traitement automatique du langage naturel et à des heuristiques. Pour évaluer FORSA, nous avons mis sur pied un benchmark adapté à la tâche de formalisation des règles métiers. Les données de ce benchmark sont issues de normes du domaine de la Construction.

Mots-clés : Règles métiers, Langage naturel contrôlé, Formalisation automatique, Web Sémantique

Abstract

This thesis focuses on automatic and semantic transformation of business rules into formal rules. These business rules are originally drafted in the form of natural language text, tables and images. Our goal is to provide to business experts a set of services allowing them to develop corpora of formal business rules. We carry out this work in the field of building engineering construction. Having formal and executable versions of the business rules enables to perform automatic compliance checking of digital mock-ups of construction projects under design.
For this we made available to business experts, the two main contributions of this thesis. The first is the development of a controlled natural language, called RAINS. It allows business experts to rewrite business rules in the form of formal rules. A RAINS rule consists of terms of the business vocabulary and reserved words such as comparison predicates, negation and universal quantification markers and literals. Each RAINS rule has a unique formal semantics which is based on the standards of the Semantic Web. The second major contribution is a service for formalization of business rules. This service implements a formalized approach proposed in this thesis and called FORSA. This service offers RAINS versions of natural language business rules submitted to it. FORSA uses natural language processing tools and heuristics. To evaluate FORSA, we have set up a benchmark adapted to the formalization of business rules task. The dataset from this benchmark are from norms in the field of Construction.

Keywords : Business rules, Controlled natural language, Automatic formalization, Semantic Web

Remerciements

Je tiens à remercier toutes les personnes qui m'ont permis de mener à bien ces années de recherche.

Je remercie le CSTB et particulièrement Patric Morand, Directeur de l'établissement de Sophia Antipolis, de m'avoir donné l'occasion d'effectuer cette thèse et d'avoir mis à ma disposition toutes les ressources nécessaires pour mener à bien mes missions. Un merci aussi à l'Université de Lyon, à l'École Doctorale Informatique et Mathématiques et au LIRIS pour m'avoir permis d'effectuer cette thèse au sein de leurs institutions.

Je remercie particulièrement Bruno, Catarina et Parisa, mes encadrants, pour leur disponibilité, leurs conseils et leur patience. Ils m'ont permis de travailler dans un climat serein et m'ont donné la possibilité de m'épanouir au cours de mes travaux de recherches. Ils ont su tout au long de ces trois années me redonner de l'énergie et structurer ma démarche quand cela s'est avéré nécessaire.

Je voudrais remercier madame Nazarenko et messieurs Bellatreche, Koukam et Le Than pour leur disponibilité et pour avoir accepté d'évaluer ce travail.

Un grand merci aux chercheurs que j'ai rencontrés et qui ont su me transmettre leur passion pour la recherche et l'innovation, et surtout qui se sont rendus disponibles. Je pense notamment à Elmar Haussman et Hannah Bast de l'Université de Freiburg, à Ricardo Usbeck et Axel Ngonga Ngomo de l'Université de Leipzig, à Christina Unger de l'Université de Bielefeld, à Luciano Del Corro du Max Planck Institute, à SungKu Kang de l'Université de l'llinois, et Saeed Hassanpour de l'Université de Stanford.

Si j'ai pu travailler sereinement au cours de ces trois années, c'est parce qu'au quotidien j'ai baigné dans un environnement propice à l'épanouissement et au bonheur au travail. Cela est dû au cadre de travail qu'offre le CSTB Sophia Antipolis et la côte d'Azur mais surtout à des collègues et notamment ceux de la "tablée" qui m'ont accueilli comme si j'avais toujours été là. Un merci des plus chaleureux à Patricia Vidal, Joëlle Cardoso

Lucas, Éric Gaduel, Bruno Fiès, Marc & Luc Bourdeau, Alain Zarli, Taylor Lum et Jonathan Lopez. Un grand merci aussi à Fabien Simon, Baptiste Gasiglia, Charly Gay, Karine Mauriaucourt, et Cécile Bensoussan pour toutes nos discussions enrichissantes. Un merci particulier à Nicolas Bus et Bruno Hilaire pour leur disponibilité sans faille. Aux dames de l'étage, notamment Christiane Pujol et Barbara Cerret.

Un merci aussi aux collègues du LIRIS et de l'Université de Lyon, en particulier Hind Benfenatki et toutes les dames qui prennent soin des doctorants : Cathérine Lombardi, Renée El Melhem, Fabienne Macro, Nathalie Roglin, Brigitte Guyader et Isabelle Buisson.

Après le CSTB, j'ai eu une deuxième demeure, ma colocation les Studines. Je tiens à remercier mes colocataires et amis Ludovic, Sabrina, Leticia, Loona, Mélissa et Annah pour tous les moments partagés et pour leur aide à la relecture de ce manuscrit.

Je ne pourrais terminer ces remerciements sans mentionner mes premiers éducateurs et toutes ces personnes qui m'ont toujours accompagné. Un merci à mes parents Papa et Mimi et à la meilleure fratrie du monde : Pif, Carole, Jimmy et Gaëlle et à tous ceux et celles qui ont agrandi cette famille depuis Yoyo, Dieudonné, Stéphanie, JM, à mes neveux, à mes "papas" et mes "mamans" et à ma grande famille. Un merci à ma chère Orline, pour sa patience et sa tolérance envers mon éloignement et mes absences. Enfin, un merci à ma deuxième famille, mes précieux amis qui ont toujours été à mes côtés malgré la distance et l'occupation.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Contents

Ce chapitre est une introduction générale à cette thèse. Il décrit le contexte dans lequel s'inscrit ce travail et résume ensuite la problématique traitée et les contributions scientifiques proposées et mises en œuvre.

1.1 Contexte et motivations

Dans le domaine de la Construction, l'importance du respect des normes est sans cesse grandissante. De la programmation, conception à la construction, exploitation et démolition, les projets de constructions sont soumis à un ensemble de réglementations strictes. Le respect de ces réglementations garantit le niveau minimum d'acceptabilité et de sécurité des bâtiments et des équipements. Les experts métiers doivent donc, pour chaque exigence mentionnée dans les normes, s'assurer qu'elle est respectée par les entités manipulées par un projet de construction donné. Cette opération porte le nom de contrôle ou vérification de la conformité.

Le contrôle de conformité requiert des experts de la Construction, des fouilles récurrentes dans le corpus de textes réglementaires. En plus d'être une opération répétitive, elle est fastidieuse et chronophage au regard de la taille et de la complexité de ce corpus. Les phases de programmation et de conception des installations étant de plus en plus réalisées à l'aide de logiciels, il devient possible d'alléger le travail des experts en automatisant le contrôle de la conformité.

Plusieurs travaux de recherches ont abordé cette problématique. On peut regrouper les résultats obtenus par ceux-ci en deux catégories principales : (*i*) La modélisation du processus de contrôle de conformité et (*ii*) la proposition de langages formels de représentation des normes. Quelle que soit la manière dont la vérification automatique de la conformité est modélisée, il est nécessaire de constituer un corpus composé d'une version formelle des normes. Par ailleurs, les experts métiers, qui sont garants de la connaissance et de la compréhension exacte des normes, doivent par conséquent les réécrire sous une forme exécutable par les ordinateurs. Or les experts métiers ne sont pas, a priori, des spécialistes en représentation des connaissances. Pour réaliser ce travail, ils s'appuient sur des personnes compétentes en matière de représentation de connaissances. Ainsi, la constitution d'un corpus de règles formelles consommables par un mécanisme de contrôle automatique de conformité, exige de l'expert en Construction un travail de réécriture ***manuel*** des normes, du langage naturel vers un langage formel, ceci avec l'***assistance d'un expert en représentation des connaissances***.

1.2 Problématique

Au regard de la taille du corpus technico-réglementaire, réécrire ***manuellement*** toutes les exigences est une opération coûteuse. De plus, ***le recours* obligatoire *à une expertise externe*** en représentation des connaissances accroît le coût en ressources humaines de cette opération. Au cours de cette thèse nous étudions ces deux problèmes.

Ces problèmes peuvent être formulés à travers la question principale suivante : ***"Comment transformer,* automatiquement*, les règles en matière de construction contenues dans les textes officiels et les textes techniques, de leur forme actuelle, i.e le* langage naturel*, en des règles* formelles *et de sorte que le résultat de cette transformation soit aisément vérifiable par les experts de la Construction (non-experts en représentation des connaissances) ?"***

Schématiquement, cette question peut se traduire par la figure 1.1.

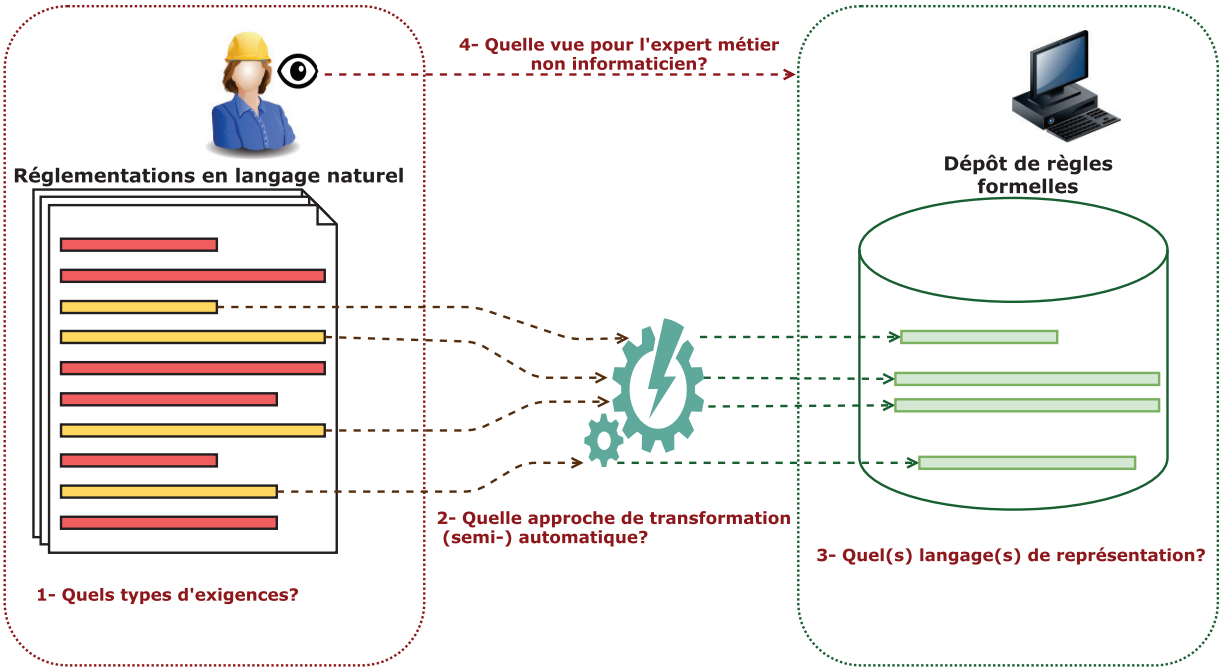


FIGURE 1.1 – Schématisation de la question principale de la thèse.

Sur cette figure on peut voir les ***facettes*** que présentent la question principale de cette thèse :

1. Quelles sont les ***caractéristiques*** des exigences réglementaires qui seront concernées par nos solutions ?
2. Quelle approche pour la transformation ***(semi-)automatique*** de ces exigences ?
3. Quel ***langage*** de représentation sera utilisé pour représenter nos règles formelles ? Comment assurer l'interfaçage avec les modèles de représentation de données dans le secteur de la Construction ?

4. L'expert métier pourra-t-il avoir ***confiance*** dans les résultats de cette transformation automatique ?
5. Comment seront ***évaluées*** nos solutions ?

1.3 Contributions

Les principales contributions de cette thèse sont :

1. Un langage naturel contrôlé, dénommé ***RAINS***, à travers lequel l'expert métier peut manipuler les règles formelles. Nous verrons que RAINS permet à son utilisateur d'écrire et de lire, indirectement, des règles formelles en toute autonomie, ce qui lui évite l'assistance d'un expert en gestion des connaissances. RAINS a été conçu pour s'adapter à des terminologies "non-naturelles" et aussi pour permettre d'accoler à chaque règle métier des informations descriptives.
2. Un service de transformation automatique des exigences du langage naturel en des règles formelles. Ce service implémente une nouvelle approche de formalisation de règles métiers, issue de notre travail de thèse. Elle porte le nom de ***FORSA***. FORSA transforme une exigence originalement en langage naturel, en une règle écrite en langage RAINS. Cette dernière est destinée à être soumise à l'appréciation de l'expert métier. Une approche de formalisation automatique de règles se doit de surmonter certains challenges identifiés dans la littérature. À ceux-ci nous ajoutons la ***détection des informations descriptives*** et l'***élagage des expressions non-pertinentes***. FORSA s'attaque à ces deux défis en plus de ceux proposés dans la littérature.
3. Un ***benchmark*** comportant un jeu de données et surtout des ***métriques*** adaptées à la tâche de formalisation des règles. C'est en s'appuyant sur ce benchmark que nous avons évaluer FORSA.

1.4 Organisation du manuscrit

Le reste du manuscrit est organisé en trois principaux temps.
En premier lieu, nous présentons le contexte de ce travail (chapitre 2). Pour cela, nous décrivons le corpus technico-réglementaire du domaine de la Construction à travers les documents qui le composent ainsi que leur organisation. Ensuite, nous présentons la modélisation des informations lors d'un projet de construction. Le chapitre se referme sur une conclusion.
Le deuxième temps de ce manuscrit est consacré à la présentation des concepts fondamentaux que nous manipulons au cours de notre travail et aussi à l'état de l'art sur

les différentes thématiques que nous abordons (chapitre 3). Ce chapitre s'ouvre par une classification des "règles" ; ce qui nous permet de préciser le type de règles métiers qui seront manipulées par RAINS et FORSA. Ensuite nous abordons les langages de représentation des connaissances. Cette partie est suivie par un état de l'art sur les approches de formalisation de règles. Enfin, nous présentons une revue de littérature sur les langages naturels contrôlés.
Les contributions de notre travail de thèse constituent le troisième temps de ce document. Nous présentons tout d'abord le langage RAINS (chapitre 4) et ensuite l'approche FORSA (chapitre 5). Les détails de nos contributions étant exposés, nous présentons le protocole d'évaluation et les résultats des évaluations de RAINS et FORSA (chapitre 6). Le manuscrit se termine par une conclusion générale et les perspectives de nos travaux (chapitre 7).

Chapitre 2

Contexte : le domaine de la Construction

Contents

Dans le domaine de la Construction, le respect de la réglementation est une contrainte de plus en plus prégnante (Grennelle I et II [1], Réglementation Thermique 2012 et 2020 à venir [2], etc.). Les experts de la Construction se trouvent régulièrement confrontés au problème de la vérification de la conformité de leurs produits et de leurs procédés, vis-à-vis de la législation locale, nationale, voire internationale. L'adoption, sans cesse croissante, d'outils logiciels pour la conception numérique de bâtiments, rend envisageable le contrôle de conformité de ces bâtiments, plus exactement de leur maquette, dès leur conception. Pouvoir effectuer cet alignement entre la maquette numérique des bâtiments et les textes technico-réglementaires requiert de disposer d'une version numérique, formelle, des prescriptions renfermées dans ces textes. Cependant, au regard de la taille et de la complexité du corpus concerné, réécrire chaque exigence légale, ***manuellement***, sous une forme exécutable est une tâche ardue et gourmande en ressources.

Le travail présenté dans cette thèse est motivé par ce constat et vise à apporter aux experts de la Construction une aide leur permettant de disposer d'exigences réglementaires formelles, issues de la législation du domaine de la Construction. L'un des objectifs majeurs de notre travail est de proposer et d'implémenter une méthode permettant de ***transformer automatiquement*** des prescriptions réglementaires, originellement en ***langage naturel*** (LN), en des ***expressions formelles***, ***exécutables***, et pouvant servir à ***détecter les points de non-conformité*** présents dans une ***maquette numérique***. Ainsi formulé, cet objectif révèle un autre aspect du problème à résoudre. Les experts de la Construction, ne possédant pas nécessairement des connaissances en langages de représentation de données, il se pose la question de la confiance de ces derniers, dans les résultats obtenus par une méthode de formalisation automatique donnée. La réponse à cette préoccupation constitue le deuxième objectif majeur de ce travail de thèse. À savoir : disposer d'une représentation des exigences réglementaires à la fois ***compréhensible par les experts métiers*** et ***strictement équivalente*** à la version formelle de ces exigences.

Dans ce chapitre, nous présentons le contexte de ce travail de thèse sous deux principaux aspects : celui de l'organisation du corpus technico-réglementaire relatif à la Construction (section 2.1) et celui de la représentation des connaissances dans le monde de la Construction (section 2.2). En guise de conclusion de ce chapitre, nous soulignons le positionnement de notre travail par rapport aux outils et pratiques existant dans le domaine de la Construction.

1. http://www.developpement-durable.gouv.fr/IMG/pdf/Grenelle_Loi-2.pdf
2. http://www.rt-batiment.fr/

2.1 Le corpus technico-réglementaire du domaine de la Construction

Les ***réglementations***, parfois appelées ***règlements***, sont un ensemble de documents juridiques dont le but est de garantir que les politiques prévues par la législation sont respectées [Yurchyshyna 2009].
Lorsque l'on limite cette définition strictement au domaine de la Construction, on peut définir les réglementations comme des règles qui précisent le niveau minimum d'acceptabilité et de sécurité pour les projets de construction, c'est-à-dire des bâtiments (tels que des maisons privées) ou des équipements (par exemple, des plate-formes élévatrices) [Satti & Krawczyk 2004].
Dans les sections qui suivent, nous nous attardons sur la manière dont sont représentées les réglementations ainsi que sur l'organisation du corpus technico-réglementaire. Dès à présent, notons que dans ce travail, nous regardons les règles de la Construction du point de vue de la législation française. Par ailleurs, nous considérons les termes "texte réglementaire", "document réglementaire" comme étant synonymes.

2.1.1 Les documents réglementaires

Dans cette section nous nous intéressons à l'organisation du contenu au sein des documents réglementaires ainsi qu'aux représentations proprement dites des exigences réglementaires.

Structure d'un document réglementaire

Les documents publics possèdent toujours une structure permettant aux utilisateurs de pouvoir exploiter convenablement leur contenu. Les textes réglementaires ne font pas exception à cette règle. Ainsi, on peut retrouver dans un document réglementaire :

- une page de garde
- un préambule ou une introduction
- un sommaire
- des parties, des annexes
- des chapitres
- des sections ou articles, des sous-sections, des sous sous-sections, etc.
- des paragraphes, des alinéas, etc.
- des notes, des commentaires

Cette structure peut varier selon qu'on ait à faire à un ***décret*** ou à un ***arrêté*** ou à une ***norme***, etc. (cf. section 2.1.2 pour les types de textes).

Par le passé, des travaux tels que ceux présentés par Lau *et al.* [2005] et Boer *et al.* [2002]

ont eu pour but de ***formater la structure*** des textes réglementaires, de sorte que celle-ci soit compréhensible par un ordinateur. Ainsi, au cours du projet E-POWER, Boer *et al.* [2002] ont mis sur pied un standard, ^{META}Lex, pour représenter l'agencement des différentes parties d'un texte. ^{META}Lex s'appuie sur un standard largement accepté, XML (eXtensible Markup Language), et est indépendant de la langue dans laquelle le texte réglementaire est écrit. Par ailleurs, ^{META}Lex permet la représentation des métadonnées pouvant être exploitées par des moteurs d'inférence pour enrichir les informations disponibles sur les documents. Toutefois, l'obtention de la représentation d'un texte respectant les spécifications de ^{META}Lex n'est pas automatique, contrairement à l'approche de représentation proposée par Lau et ses collègues en 2005. Ces derniers proposent des ***patrons*** (*patterns*) pour une extraction automatique de la structure des documents réglementaires. Ces patrons sont conçus pour reconnaître les différentes subdivisions d'un document et les représenter de manière formelle sous la forme d'une arborescence s'appuyant sur le standard XML. Cependant, cette méthode de formalisation automatique de la structure d'un texte réglementaire nécessite plusieurs adaptations pour pouvoir s'attaquer à la totalité d'un corpus technico-réglementaire. En effet, comme mentionné au terme du paragraphe précédent, tous les types de textes n'ont pas la même organisation. De plus, les différentes subdivisions d'un texte ne sont pas signalées de manière uniforme : la numérotation (par exemple, 4.1 ***vs*** IV.I ***vs*** 4-1, etc.), le label de la subdivision (par exemple, section 1 ***vs*** *I*. ***vs*** §1, etc.), varient.

Nous voyons, que les textes réglementaires possèdent des structurations diverses. En outre, des efforts sont faits par les chercheurs pour extraire automatiquement et représenter de manière formelle cette structure. La question de la représentation des subdivisions des documents réglementaires, de manière exploitable par un ordinateur, n'est pas un point focal de notre travail de thèse. Nous nous intéressons, lorsque nous parlons de formalisation des exigences ou des contraintes réglementaires, uniquement à ce qui est énoncé dans telle ou telle autre partie d'un document donné. Par exemple, si nous considérons la prescription ***"Les accès à la gaine entièrement close doivent être protégés par des portes palières."*** se trouvant à section 5.8.1 du document [NF EN 81-41 2011], notre but est de savoir comment réécrire de manière formelle cette exigence ; le numéro de la section, de la page, du chapitre concerné, sont des métadonnées qui n'entrent pas dans la formalisation, proprement dite, de cette exigence.

Représentations des exigences réglementaires

En conclusion de la section précédente, nous insistions sur le fait que notre travail s'intéressait principalement aux exigences réglementaires proprement dites, et peu à la position de ces exigences au milieu des subdivisions des documents. Dans cette section, nous présentons les différentes formes sous lesquelles on peut retrouver les contraintes dans les textes réglementaires, à savoir les ***textes***, les ***tableaux*** et les ***images***. Les exi-

gences permettant d'illustrer nos propos proviennent toutes de la norme NF EN 81-41 [2011] - cette norme a été choisie au hasard parmi les normes relatives à un ou plusieurs aspects des projets de construction.

1. **Textes**
   Dans les documents réglementaires, il est habituel de rencontrer des prescriptions exprimées sous une forme textuelle. Aussi, des exigences peuvent être transcrites :
   — À l'aide d'***une seule phrase***. Exemple : *"La vitesse de la plate-forme ne doit pas dépasser 0,15 m/s"*.
   — En se servant de ***plusieurs phrases***. Exemple : *"Les parois de la gaine doivent pouvoir résister à un effort de* 300 *N appliqué perpendiculairement en tout point sur une surface de* 5 cm^2 *de forme ronde ou carrée, sans présenter de déformation élastique supérieure à* 15 *mm et sans présenter de déformation permanente. Toutefois, la déformation élastique des parois de la gaine ne doit pas dépasser le jeu de fonctionnement entre la plate-forme et les parois de la gaine"*. Nous pouvons constater que les deux phrases doivent nécessairement être considérées simultanément pour pouvoir comprendre la prescription.
   — Par le biais de ***listes***. Exemple : *"Le réservoir doit être conçu et construit de façon à :*
   a. faciliter la vérification du niveau de fluide hydraulique dans le réservoir ;
   b. faciliter le remplissage et la vidange."
2. **Tableaux**
   Pour éviter une hyper verbosité, structurer des idées à l'aide de tableaux permet dans certains cas de comprendre clairement le message énoncé. De plus, les tableaux peuvent permettre d'éviter la répétition et faciliter la mise en correspondance ou la comparaison de plusieurs éléments. C'est ainsi qu'on peut retrouver des tableaux dans les documents réglementaires. Le tableau 2.1 extrait de la norme NF EN 81-41 [2011] montre un exemple d'exigences sous forme tabulaire.

Tableau 2.1 – Exemple d'exigences réglementaires présentées sous forme de tableau. Extrait de la norme NF EN 81-41 [2011] (Tableau 2 : Dimensions utiles minimales de la plate-forme).

Utilisation principale	**Dimensions utiles minimales (largeur × longueur)**	**Charge nominale minimale (kg)**
Fauteuils roulants de types A et B avec un accompagnateur et des services en angle	1100 × 1400	385
Fauteuils roulants de types A et B avec un accompagnateur	900 × 1400	315
Usager seul, debout ou en fauteuil roulant de type A	800 × 1250	250

3. **Images**
   En plus des formes textuelles et tabulaires, on peut retrouver dans des documents réglementaires des images accompagnant des exigences. Dans ces cas, les schémas

et autres figures servent généralement d'aide à la compréhension. Un schéma peut ainsi :

— Représenter l'ensemble des détails de l'exigence réglementaire. Exemple : *"Le dispositif de charge des batteries doit correspondre à la Figure 4a pour une charge en courant alternatif"*. la référence "Figure 4a" désigne une figure de la norme NF EN 81-41 [2011]. Pour illustration, nous la reproduisons à la figure 2.1.

— Illustrer un objet mentionné dans le document. Exemple : *"**5.6.1 Généralités** Voir exemple à la Figure 5"*. De manière identique à l'exemple précédent, nous reproduisons la "Figure 5" de la norme NF EN 81-41 [2011], par la figure 2.2.

— Expliciter une exigence textuelle en reprenant ses éléments normatifs sur ce schéma. Exemple : *"Aucune cavité ni saillie dans les surfaces intérieures des parois de la gaine ne doit dépasser 5 mm et les saillies de plus de* 1.5 *mm doivent être chanfreinées avec un angle d'au moins 15° par rapport à la verticale (voir Figure 6)."*. La "Figure 6" de la norme NF EN 81-41 [2011], est reprise dans la figure 2.3.

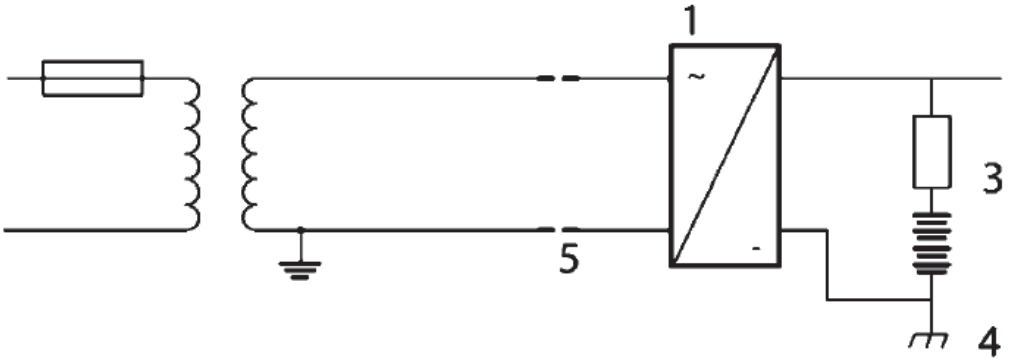


Figure 2.1 – Reproduction de la "Figure 4a" de la norme NF EN 81-41 [2011]. ***Titre original :*** Alimentation de charge pour les plates-formes élévatrices alimentées par batterie (les légendes du schéma original n'ont pas été reproduites ici)

Au terme de cette section, nous pouvons nous rendre compte de la multitude de manières qu'il y a de formuler des prescriptions dans un document réglementaire. Au cours de ce travail de thèse, la méthode de formalisation que nous présentons concerne uniquement les ***exigences réglementaires exclusivement sous une forme textuelle***. Nous avons distingué trois types d'exigences sous la forme de texte : celles dont le sens est complètement circonscrit par une phrase, celles étant exprimées par plusieurs phrases ainsi que celles qui sont spécifiées à l'aide de listes de phrases. Parmi ces trois catégories de représentations textuelles de contraintes réglementaires, seule la première, c'est-à-dire ***les prescriptions dont le sens se limite à une phrase***, fera l'objet de notre service de formalisation. En effet, les exigences sur plusieurs phrases requièrent des mécanismes complexes (résolution de co-références, explicitation des exceptions, etc.) pour pouvoir disposer de la totalité de leur sémantique. Par ailleurs, en se concentrant sur les exigences réglementaires textuelles, nous renonçons à formaliser automatiquement les semi-structures que sont les tableaux. Il n'existe pas une manière unique de disposer les

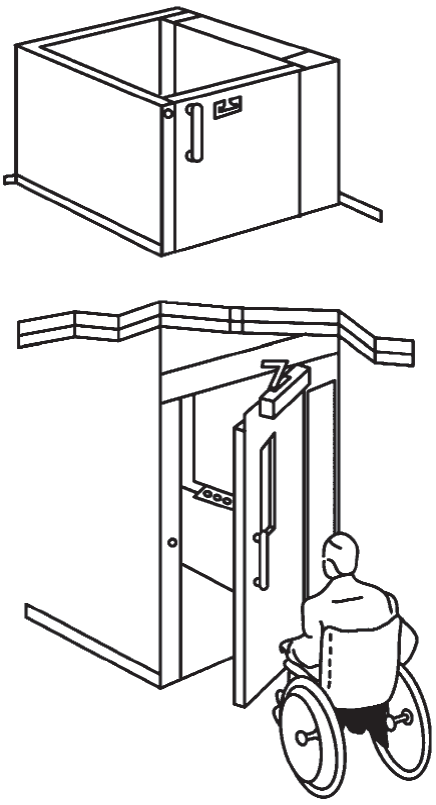


FIGURE 2.2 – Reproduction de la "Figure 5" de la norme NF EN 81-41 [2011]. ***Titre original :*** Exemple de plate-forme élévatrice verticale avec gaine entièrement close

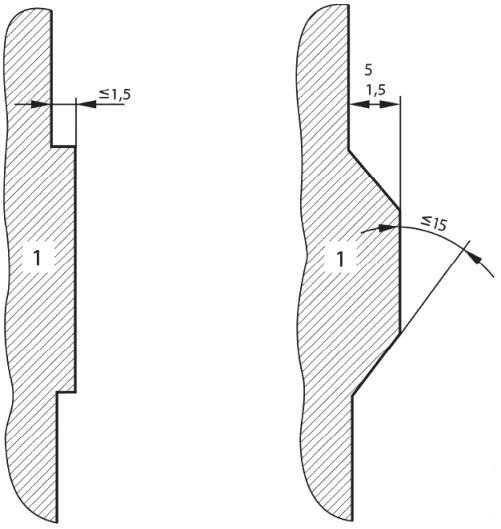


FIGURE 2.3 – Reproduction de la "Figure 6" de la norme NF EN 81-41 [2011]. ***Titre original :*** Dimensions des saillies admises pour la gaine entièrement close

entrées d'un tableau. Certains peuvent être de simples matrices et d'autres posséder des sous-cellules, voire des sous sous-cellules, etc. De plus, le contenu d'une celulle n'est pas toujours un texte ou une série de nombres, mais peut être une image, une formule mathématique, des listes pouvant disposer de sous listes. Aussi, les interprétations de figures et autres schémas à la recherche d'exigences réglementaires potentielles ne seront pas de possibles entrées de notre service de formalisation.

2.1.2 Volume et organisation du corpus technico-réglementaire

Le domaine de la Construction est régi par de nombreux textes ayant une organisation bien déterminée. D'après le référentiel réglementaire, dénommé ***REEF***[3], mis en place et régulièrement mis à jour par le Centre Scientifique et Technique du Bâtiment (CSTB), on estime à environ 4511 le nombre de documents qui constituent le corpus technico-réglementaire du domaine de la Construction. Ces documents remplissent près de 76.000 pages et ont des valeurs juridiques diverses, que nous présentons ci-après.

3. http://boutique.cstb.fr/Product/reef-classique

Hiérarchie des textes réglementaires dans le contexte Français

Bien que le but de notre travail n'est pas de concevoir un système de gestion de règles métiers (***Business Rules management System*** (BRMS)), il est important de décrire l'organisation des textes réglementaires en France. En effet, les exigences que nous écrivons sous une forme processable, serviront plus tard pour des opérations de contrôle automatique de la conformité. Comme cela a été souligné, notamment dans les travaux de thèse de Yurchyshyna [2009] :

— Cette classification réglementaire est un point de départ pertinent pour une organisation informatisée des règles dans un dépôt.
— Cette hiérarchie ainsi que les métadonnées qui l'accompagnent, permettent d'enrichir, d'annoter, les exigences réglementaires proprement dites.
— Pour un produit donné, la manière dont le processus de vérification de la conformité est effectué, peut s'appuyer sur cette organisation. En effet, tous les types de textes réglementaires n'ayant pas la même pertinence, la même force dans un contexte donné, il est parfois nécessaire de définir l'ordre dans lequel il faudra faire appel aux prescriptions extraites de ces textes.

La classification des textes réglementaires présentée ici, reprend le travail exposé dans [CSTB 2002], et rédigé par les experts du CSTB.

Tout d'abord, notons que les documents réglementaires peuvent avoir une portée nationale (française) ou européenne. Les textes nationaux sont valables sur l'ensemble du territoire Français alors que les textes européens s'appliquent à l'Union Européenne et donc, aussi à la France. Dans certains cas, les normes européennes disposent d'équivalents stricts dans la réglementation Française. C'est le cas de la norme qui nous sert d'illustration pour les exemples mentionnés dans ce manuscrit, la norme NF EN 81-41. Son équivalent européen est la norme EN 81-41.

La figure 2.4 présente une classification des textes Français. Ils se distinguent en deux groupes : les textes ***officiels*** et les textes ***techniques***. Chacun se subdivisant en des catégories plus petites.

— Les textes officiels peuvent être ***obligatoires*** ou à caractères ***informatifs***. Comme leurs noms l'indiquent, chaque exigence présente dans un texte à caractère obligatoire doit impérativement être vérifié. Par contre, le contenu d'un texte informatif est un ensemble de recommandations et de bonnes pratiques.
  — Parmi les textes officiels obligatoires, on retrouve les ***codes***, les ***lois***, les ***décrets*** et les ***arrêtés***.
  — Au nombre des textes à caractères informatifs, nous avons les ***circulaires*** et les ***questions écrites***.
— Les textes techniques sont de deux genres : les ***textes concernant les produits, les procédés, les équipements*** et les ***règles de mise en oeuvre***.
  — Les ***normes*** (Française NF, Européenne EN, normes ISO [4]), les ***Avis Techniques***

4. Normes de l'Organisation internationale de normalisation.

(Atec), les ***Documents Techniques d'Application*** (DTA) et les ***Appréciations Techniques d'Expérimentation*** (ATEx), composent les textes concernant les produits, les procédés, les équipements.

— Les règles de mise en oeuvre sont formées par les ***Documents Techniques Unifiés*** (DTU), les ***Normes-DTU***, les ***Règles de Calcul-DTU***, les ***Règles de Calcul***, les ***documents de mise en oeuvre de produits, procédés, équipements relevant des Atec*** et les ***Règles ou recommandations professionnelles***

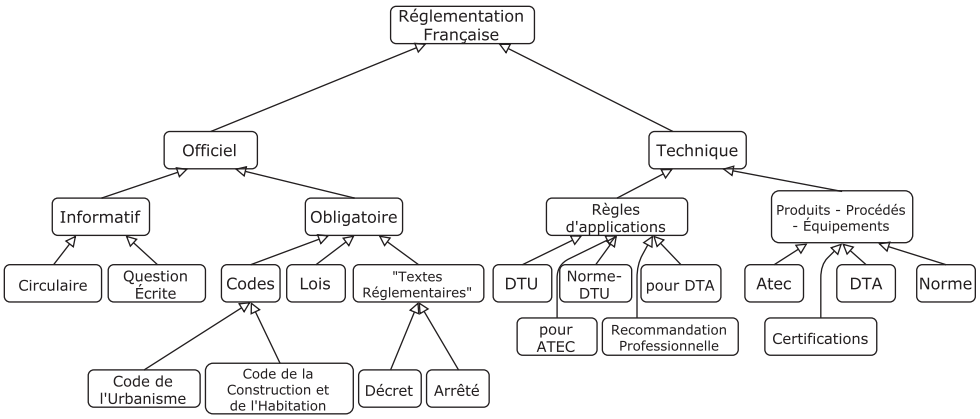


FIGURE 2.4 – Classification des textes réglementaires Français (basée sur CSTB [2002]).

Maintenant que nous avons présenté l'organisation du corpus technico-réglémentaire, nous nous attardons sur la manière dont les informations relatives aux projets de construction, sont modélisées.

2.2 Modélisation des Données du Bâtiment

2.2.1 Gestion des données du Bâtiment

Les projets de construction (tels que les habitations individuelles, les installations industrielles, les ponts, etc.) sont conçus à travers des cycles de collaboration entre des architectes et des ingénieurs. Ces dernières décades ont vu se généraliser l'usage d'outil informatique grâce à la multiplication des logiciels de conception (en anglais *Computer-Aided Designs tools* (CAD)), dédiés à la construction. La notion de maquette numérique ou BIM (Building Information Modelling) fait son apparition dans les années 1970 [Eastman *et al.* 2011]. À la question "Qu'est ce que le BIM ?", Le National BIM Standard, institution américaine, répond en disant ***"Le BIM est une représentation numérique des caractéristiques physiques et fonctionnelles d'une installation. C'est une ressource favorisant le partage de connaissances et qui permet d'obtenir des informations sur une installation, formant une base fiable pour les décisions au cours du cycle de vie de celle-ci ; ledit cycle allant de la conception de l'installation, à sa démolition."*** [5] [NBIMS-US^{TM}

5. Traduit de l'anglais.

2016]. Ainsi, pour rester proche du crayon et du papier, le BIM en tant que représentation des bâtiments, inclut les informations géométriques. À celles-ci, s'ajoutent les relations spatiales, les informations géographiques, les quantités ainsi que les caractéristiques des composants du bâtiments (par exemple les informations sur les fournisseurs) [Lee *et al.* 2006]. Par ailleurs, en tant que modèle de gestion complet des informations relatives à la vie des installations, le BIM intègre les notions de temps et de coûts. Ainsi, un des principes de base du BIM est la collaboration, entre les parties prenantes d'un même projet, à différentes phases du projet. Les collaborateurs peuvent par conséquent, à tout moment, ajouter, extraire ou mettre à jour les informations du BIM [NBIMS-US^{TM} 2016]. La collaboration, entre les parties prenantes, que favorise le BIM, pose l'épineuse question de l'***interopérabilité***. En effet, il faudrait que les différents produits logiciels, utilisés par les collaborateurs d'un projet, puissent communiquer "sans restriction d'accès ou de mise en oeuvre" -selon la définition de l'interopérabilité [AFUL 2016]. C'est ainsi qu'une volonté est née des acteurs du bâtiment, afin de rendre possible et de faciliter les échanges entre les différentes applications utilisées par les corps de métier, hétéroclites (architectes, bureaux d'étude, maîtres d'ouvrage, etc.). Cet effort a donné naissance à un standard ouvert, les ***IFC*** (*Industry Foundation Classes*).

2.2.2 Industry Foundation Classes (IFC)

La norme ISO 16739 :2013 [6] fournit un schéma conceptuel de données et un format de fichier pour les échanges d'informations sur le BIM, le format IFC. Le modèle de données IFC est spécifié en langage ***EXPRESS*** [7], ce dernier conforme à la norme ISO 10303-11 (STEP [8] part 11). Le modèle IFC est un modèle ***orienté objet*** qui définit des classes associées à tous les objets de construction. Un bâtiment représenté en IFC, est donc en réalité, un ensemble hiérarchisé d'instances de classes définit dans le modèle IFC. Par exemple, la hiérarchisation d'un bâtiment s'effectue ainsi :

- Un ***bâtiment*** contient plusieurs ***étages***
- Un ***étage*** contient plusieurs ***salles***
- Une ***salle*** contient des ***ouvertures***
- Une ***ouverture*** peut être occupée par une ***porte***, une ***fenêtre***, etc.

Un projet en IFC peut être représenté par un fichier respectant le modèle de données EXPRESS ou alors sous un fichier respectant la spécification IfcXML [9].
À la section 3.2 consacrée à la représentation des connaissances, nous aborderons avec

6. http ://www.iso.org/iso/catalogue_detail.htm ?csnumber=51622
7. http ://www.iso.org/iso/fr/iso_catalogue/catalogue_tc/catalogue_detail.htm ?csnumber=38047
8. Le *STandard for the Exchange of Product model data* est un standard ISO (10303) dont le but est de fournir une représentation de l'information relative au produit ainsi que les mécanismes et définitions nécessaires à l'échange de données de produit. `http://www.iso.org/iso/fr/catalogue_detail?csnumber=20579`
9. Comme son nom l'indique, un fichier .IfcXML est un fichier IFC utilisant la structure XML. Le format IfcXML est conforme à la norme ISO10303-28 qui décrit la représentation XML des schémas de données EXPRESS.

plus de détails la représentation des données au format EXPRESS, et par ce biais, la représentation des données en IFC.

2.2.3 Outils de Manipulation des IFC

De nombreux outils sont dédiés à la manipulation des fichiers IFC. On peut classer ces outils selon les opérations qu'ils permettent d'effectuer sur un fichier IFC. Ainsi on retrouve [Yurchyshyna 2009] :

— Des outils de ***visualisation*** (*viewers*) qui permettent de visualiser un modèle IFC, la structure d'un projet ou encore les propriétés des objets contenus dans un modèle.
— Des ***éditeurs*** (*IFC text browsers*) dont le but est de présenter le contenu du fichier IFC d'une manière plus conviviale, à des fins de débogage.
— Des outils de ***conversion*** (*converters*) qui servent à transformer un fichier du format IFC vers un ou plusieurs autres formats
— Des ***correcteurs*** (*syntax checkers*) qui sont des outils qui permettent de s'assurer de la validité des fichiers .ifc au regard de la syntaxe formelle du langage IFC.

Tableau 2.2 – Quelques outils de manipulations des fichiers IFC. Extrait de [Yurchyshyna 2009, p. 21].

Catégorie	**Outil (Institution)**	**Brève description**
Visualisation	DDS IFC Viewer (de Data Design System)	*Viewer* pour des fichiers IFC
	IfcViewer (de Forschungszentrum Karlsruhe)	*Viewer* pour des fichiers IFC
	Nemetschek IFC Viewer (de Nemetschek AG)	3D IFC *viewer*, qui supporte les format IFC et IfcXML
	Solibri Model Viewer [10] (de Solibri)	Visualisation des IFC et *model checking* (détection des défauts, mise en évidence des incohérences)
	Ifc Engine Viewer (par le TNO Building Research)	3D *viewer* pour les IFC
Édition	IfcQuickBrowser (de GEM Team Solutions)	Éditeur pour de gros fichiers IFC. Les fichiers sont présentés sous la forme d'une arborescence
Conversion	CIS/2 to IFC Translator (du NIST)	Convertisseur pour les modèles de produits pour les structures en acier
	IFC to Excel Translator (du NIST)	Permet la création d'un fichier Excel à partir d'un fichier IFC
	EXPRESS to OWL Converter [Pauwels & Terkaj 2016]	Permet de convertir les fichiers EXPRESS en des fichiers OWL
Correction	IfcObjectCounter (du Forschungszentrum Karlsruhe)	Correcteur de fichiers IFC

Parmi les outils présentés dans le tableau 2.2, un seul intègre des fonctionnalités de détection de non-conformités. Il s'agit de l'outil de visualisation *Solibri Model Viewer* qui

10. http ://www.solibri.com/products/solibri-model-viewer/

intègre un *model ckecker*. Ce *checker* permet ***dès la phase de conception*** de pouvoir détecter des défauts et donc d'économiser des coûts et du temps d'une éventuelle réparation si les installations étaient construites avec des défauts - dont ceux détectés par l'outil. Malheureusement, les règles insérées dans ces outils sont formalisées, ***à la main***, une par une, par les experts métiers assistés de personnes ayant des compétences en représentation formelle des connaissances.
Néanmoins, pouvoir anticiper des malfaçons et autres anomalies dès la phase de conception, et donc bien avant l'existence concrète des constructions, ***valide la démarche de l'utilisation du BIM*** pour la gestion des projets de Construction. Par ailleurs, aujourd'hui, disposer d'exigences formelles en accord avec la réglementation en matière de Construction, nécessite soit une double compétence d'expert métier et de logicien, soit de faire collaborer deux experts, chacun ayant les compétences adéquates. Dans ce travail de thèse, nous proposons de contourner la nécessité de disposer d'un logicien, laissant ainsi la place à un expert métier pouvant, de manière autonome, obtenir des règles formelles à partir d'exigences dans leur représentation originale.

2.3 Conclusion du Chapitre

Nous avons consacré ce chapitre à la présentation du contexte d'application de notre travail de thèse, qui est le domaine de la Construction. Nous avons brièvement parlé des généralités inhérentes au secteur de la Construction pour nous concentrer sur l'aspect réglementaire de ce secteur, en nous plongeant dans le corpus des textes de lois. Nous avons ainsi pu présenter l'organisation des textes réglementaires en France, et fait ressortir la structure complexe des documents réglementaires, c'est-à-dire la subdivision du document en parties, chapitres, sections, etc. En outre, il a été question de la manière dont les exigences sont exprimées. Ces dernières peuvent se présenter sous une forme textuelle (phrases, listes, etc.), sous la forme d'un tableau ou encore d'un schéma. Dans le travail de formalisation automatique d'exigences réglementaires qui est le nôtre, nous nous intéressons uniquement aux exigences exprimées à l'aide d'une unique phrase. Nous verrons dans le chapitre suivant qu'une multitude de règles répondent à cette contrainte et qu'il faut relever de nombreux défis pour parvenir à une approche de formalisation automatique de règles.

En deuxième partie de ce chapitre, nous avons fait un bref exposé sur la gestion des données dans le monde de la Construction. Cela nous a permis de parler de la gestion numérique des projets de Construction, qui est supportée par le concept de *Building Information Modelling* (BIM). Le BIM est une représentation multi-dimensionnelles des caractériqtiques d'une infrastructure. Il permet de prendre en compte, en plus de la

géométrie et des informations quantitatives des installations, les notions de temps et de coûts. De plus, pour faciliter les échanges de données entre les différents collaborateurs sur un même projet-BIM, un standard ouvert a été proposé pour représenter les données, le standard IFC (*Industry Foundation Classes*). IFC est un modèle de données orienté objet et spécifié dans le langage EXPRESS. Ainsi, représenter un bâtiment en IFC, revient à instancier la hiérarchie des classes disponibles dans le modèle IFC. Pour clore ce chapitre, nous nous sommes intéressés aux outils actuels, qui permettent de manipuler les fichiers IFC, ainsi qu'aux tâches que ces outils servent à réaliser. Cela a permis de nous rendre compte que le contrôle de conformité, ou la détection de défauts, sont des fonctionnalités très peu proposées par les outils de manipulation des IFC. Si la correction des défauts de construction, dès la phase de conception, présente un avantage indéniable, il faut néanmoins pouvoir disposer des prescriptions réglementaires sous une forme processable. Cette condition se heurte à deux difficultés majeures. La première est le volume et la complexité de l'organisation des textes réglementaires. La seconde est la nécessité de posséder à la fois, la connaissance métier et la compétence en représentation des connaissances, pour pouvoir mener à bien cette mission. C'est ainsi que le but de notre travail de thèse est d'aider à tout ou partie du processus de formalisation des exigences réglementaires, en supprimant la nécessité de recourir à un expert logicien pour la représentation des exigences sous une forme processable.

Chapitre 3

Concepts Fondamentaux et État de l'Art

Contents

Dans ce chapitre, nous présentons les notions fondamentales qui nous permettent d'exposer les résultats de ce travail de thèse proprement dit. Par ailleurs, ce chapitre est le lieu de présenter des travaux visant des objectifs similaires aux divers aspects que revêt notre travail ; état des lieux nécessaires pour cerner les apports de ce travail de thèse. L'organisation de ce chapitre est la suivante :

— Une première partie (section 3.1) dédiée à la classification des règles, qui synthétise le regard porté par les logiciens sur les règles.
— Une seconde partie (section 3.2) qui est le lieu pour nous de présenter des langages de représentation formelle des connaissances.
— Ensuite vient une troisième partie dans laquelle nous présentons les méthodes existantes de réécriture automatique de règles en LN en des règles formelles (section 3.3).
— Enfin, un état de l'art sur les modèles de représentation de connaissances pouvant servir de pont entre l'humain et l'ordinateur (section 3.4) clôt notre série de présentations de concepts fondamentaux et des travaux antérieurs.
— Une synthèse générale referme ce chapitre.

3.1 Classifications des règles

Dans le chapitre précédent, nous avons présenté les exigences réglementaires d'un point de vue guidé par le domaine de la Construction. Ces exigences étant des règles - au sens de la ***logique*** [1] - nous nous intéressons à la manière dont les règles sont organisées par les logiciens.

3.1.1 Classification générale des normes

Dans son travail de pionnier sur les normes, Von Wright [1963, chap. 1] distingue trois principales (parmi six) catégories de normes.

1. Les ***règles*** ou ***descriptions*** qui donnent vie à des concepts ou des activités. Sans ces règles, ces concepts et activités ne pourraient "exister". Ces règles sont ainsi qualifiées de ***constitutives*** (“***constitutive rules***”) ou ***déterminantes*** (“***determinative rules***”) [Gordon *et al.* 2009]. C'est le cas des règles d'un jeu, ou des règles de la logique et des mathématiques [Von Wright 1963].
2. Les ***presprictions*** se composent des ***obligations***, des ***permissions*** et des ***prohibitions*** formulées par une autorité à une personne en position de sujet Von Wright [1963]. En général, les prescriptions indiquent au sujet, les actions prévues et leurs conditions d'application et la nature de leur orientation Gordon *et al.* [2009].

1. La logique peut se définir comme l'étude des règles formelles que doit respecter toute argumentation correcte [Dubucs 2005].

3. Les ***directives*** encore appelées ***règles techniques*** qui permettent de lister un ensemble d'actions à mener dans l'atteinte d'un certain but. Exemple : ***"Les éléments constitutifs (l'engrenage de l'unité d'entraînement, par exemple) doivent être protégés pour éviter les risques de blessures aux personnes."***

Dans le contexte de ce travail, à savoir la formalisation des règles pour le contrôle de conformité, on peut réduire cette classification. En effet on pourrait considérer nos règles comme étant soit des ***prescriptions strictes***, soit des ***recommandations***. Une telle classification nous est présentée dans les travaux de Yurchyshyna [2009] :

— Une prescription stricte décrit des exigences qui doivent obligatoirement être respectées. Elle présente, les critères, les directives techniques et les solutions à des problèmes précis. Exemple : ***"Toutes les roues dentées d'entraînement doivent être en métal et posséder au moins 16 dents découpées mécaniquement."***

— Une recommandation, aussi appelée norme orientée performance, définit des exigences optionnelles, sous forme de conseil. Les recommandations sont en quelque sorte des bonnes pratiques. Exemple : ***"Il est recommandé d'utiliser la plateforme au moyen de boutons-poussoirs classiques, de manettes ou de dispositifs similaires"***.

Rappelons que le but du contrôle de conformité est de relever les éléments de la maquette d'un produit, qui ne respectent pas la réglementation. Ainsi défini, une maquette numérique ne peut présenter des non-conformités que vis-à-vis des exigences qui doivent impérativement être respectées. Ainsi, les recommandations, à moins de les "transformer" en prescription stricte, ne peuvent permettre d'identifier des défauts réglementaires dans une maquette numérique.

3.1.2 Hiérarchie des règles d'après le *Rule Markup Initiative*

Indépendamment du fait qu'une norme commande ou recommande, on pourrait aussi classifier les règles en fonction de leur ***structure*** intrinsèque, c'est-à-dire la manière dont ses composants sont agencés. Dans le travail de spécification du langage de règle RuleML [2] (Rule Markup language), il nous est proposé une hiérarchie des règles basées sur leur structure [3]. Cette taxonomie permet de disposer de façon immédiate, de la liste des différents types de règles depuis les ***règles de dérivations*** (*derivation rules*) , aux ***règles de réaction*** (*reactions rules*) en passant par les ***règles de transformation*** (*transformation rules*) [Boley *et al.* 2010][Gordon *et al.* 2009]. Cette hiérarchie est graphiquement représentée sur la figure 3.1.

Règles de dérivations
Les règles de dérivations (*derivation rules*) désignent les implications et les règles d'inférences [Boley *et al.* 2001].

2. `ruleml.org`
3. Nous présentons le langage RuleML proprement dit à la section 3.2.2.

Logique modale
La logique modale est une logique qui permet de formaliser des éléments de modalités, c'est-à-dire de spécifier des qualités du vrai [Dimaresis 2007]. La logique modale est, à proprement parler, l'étude des inférences à partir des expressions telles que "il est nécessaire" et "il est possible que" [Dimaresis 2007].

Règles de délibérations
Les règles de délibérations (*deliberation rules*) réunissent les règles modales et les règles de dérivations [Boley *et al.* 2001].

Règles de réactions
Les règles de réactions (*reaction rules*) sont relatives à l'invocation d'actions en réponse aux événements qui nécessitent la prise d'une décision par un sujet [Paschke & Boley 2010].

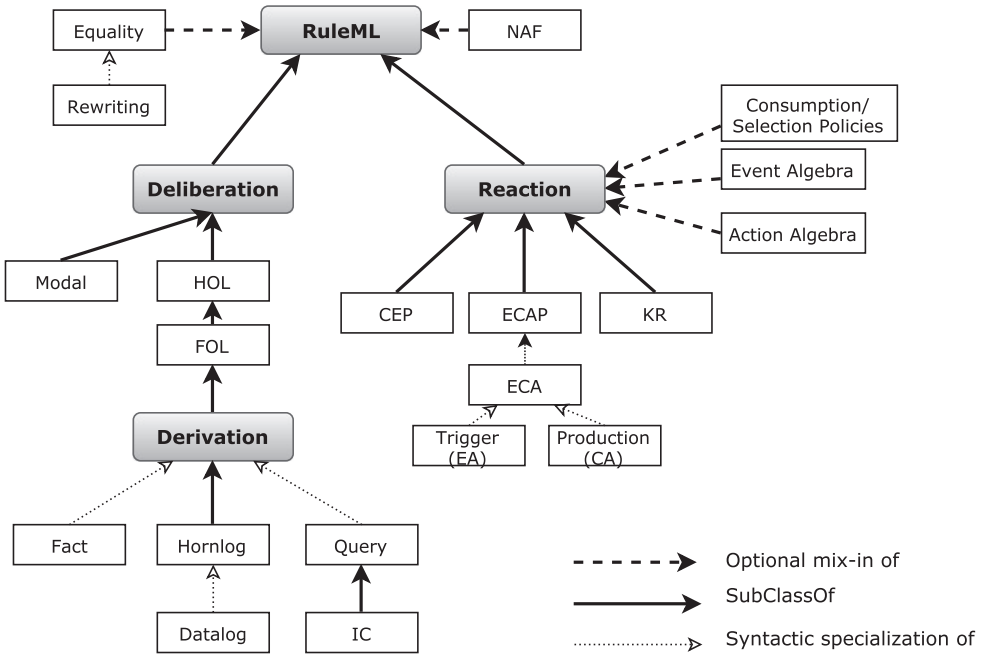


FIGURE 3.1 – Hiérarchie des règles proposée par RuleML [Boley *et al.* 2010].**Naf** : Negation as failure. **HOL** : High Order Logic. **FOL** : First order Logic. **IC** : Integrity Constraints. **KR** : Knowledge Representation. **CEP** : Complex Event Processing. **ECA** : Event Condition Action. **ECAP** : ECA Postcondition. **EA (trigger)** : Event Action (no condition). **CA (Production Rules)** : Condition Action (no Event)

En tant que support pour un processus de contrôle de conformité, les règles auxquelles nous allons nous confronter sont des règles de délibérations au sens de RuleML. En effet, des règles devant nécessiter une action, comme dans le cas des *reaction rules*, ne peuvent

pas, *stricto sensu*, permettre de dire si oui ou non, une maquette présenterait un quelconque défaut. Par exemple, la prescription suivante ***"L'actionnement de ces dispositifs (bords sensibles, cellules photoélectriques ou barrières immatérielles) doit arrêter la plate-forme élévatrice avant que des parties rigides n'entrent en contact forcé"***, ne peut pas, en l'état être utilisée pour vérifier la conformité d'une plate-forme élevatrice. Par contre, en fonction de la modélisation de la plate-forme, cette contrainte réglementaire pourrait permettre de se poser la question de savoir si une plate-forme donnée dispose de "bords sensibles", de "cellules photoélectriques" ou "barrières immatérielles" et si ses dispositifs sont "actionnables". Une réponse négative à l'une de ces questions serait alors interprétée comme une non-conformité. Par contre, des réponses positives ne pourraient pas amener à conclure à une conformité de la plate-forme vis-à-vis de cette prescription. Pour cela, il faudrait, dans un cas pratique, vérifier que ces dispositifs sont actionnables et permettent effectivement d'arrêter le mouvement de la plate-forme dans les conditions mentionnées.

3.1.3 Classification basée sur les types de "normalisation" des règles

La ***normalisation*** est une transformation dont le but est de clarifier un texte en ayant recours à des termes de référence (c'est-à-dire pris dans un dictionnaire) et en explicitant les informations implicites. En vue de sa formalisation, un texte normalisé présente moins de difficultés (que le texte original) pour une personne ayant des compétences en logique et en représentation des connaissances. Cette notion a été introduite par Guissé *et al.* [2012] dans leurs travaux sur l'acquisition et la maintenance des règles métiers à partir des textes réglementaires. La normalisation implique une reformulation d'une règle, possiblement ambiguë ou difficile à comprendre pour les non-experts du domaine, tout en préservant son sens original. Guissé *et al.* ont identifié quatre (04) principales catégories de normalisation. Chaque catégorie se décompose à son tour en sous-catégories. En nous appuyant sur le travail présenté dans [Guissé *et al.* 2012], nous présentons brièvement les différents types de normalisation. Pour chaque type de formalisation nous présentons des exemples validés par les experts métiers.

La normalisation lexicale

La normalisation lexicale consiste à remplacer les termes de l'exigence originale par d'autres issues d'un vocabulaire contrôlé. Nous employons le terme vocabulaire contrôlé pour désigner toute ressource regroupant les termes du domaine d'étude. Ainsi, une ***ontologie de domaine***, un ***thésaurus***, voire un ***dictionnaire*** peuvent jouer le rôle de vocabulaire contrôlé.
Exemple :

- **Règle originale :** *"La* **vitesse** *de la* **plate-forme** *ne doit pas dépasser 0,15 m/s."*
- **Règle après normalisation lexicale :** *La* **vitesse nominale** *de la* **plate-forme élévatrice** *ne doit pas dépasser 0,15 m/s.*

Dans cet exemple, nous voyons que l'exigence réglementaire telle qu'écrite dans la norme mentionne les termes ***"vitesse"*** et ***"plate-forme"***. La normalisation lexicale, s'appuyant sur le glossaire de la norme [NF EN 81-41 2011, section 3], remplace ces deux termes par leur forme normalisée qui sont respectivement ***"vitesse nominale"*** et ***"plate-forme élévatrice"***.

La décontextualisation

Aussi connue sous le nom de ***résolution de co-référence*** (*coreference resolution*), la décontextualisation vise à expliciter et préciser une référence implicite. Ladite référence implicite peut être :

1. Un ***pronom*** ou un ***adjectif possessif***. **Exemple :** *"Un manomètre doit être prévu.* **Il** *doit être raccordé au circuit entre le clapet de non-retour ou la (les) soupape(s) de descente et le robinet d'isolement."* Le pronom ***"il"*** renvoie au ***"manomètre"*** de la prescription précédente.
2. Un ***terme générique***. **Exemple :** *"Un* **dispositif d'arrêt d'urgence** *conforme à l'EN ISO 13850 doit être installé sur la plate-forme. Lorsqu'il est actionné, ce* **dispositif** *doit directement interrompre la chaîne électrique de sécurité. Ce* **dispositif** *doit être clairement visible et accessible à l'usager et doit être facile à manoeuvrer."*. Parmi ces trois phrases que nous venons de reproduire, le terme ***"dispositif"*** apparaît dans les deux dernières phrases sans précision. Le ***"dispositif"*** dont il est question est le dispositif d'arrêt d'urgence, tel que mentionné par la première phrase. Plusieurs "dispositifs" étant évoqués dans la norme ("dispositif électrique", "dispositif de sécurité", "dispositif de surcharge", etc.), la décontextualisation permet de lever cette ambiguïté.
3. Une ***référence croisée***. **Exemple :** *"Un dispositif d'arrêt d'urgence conforme à l'**EN ISO 13850** doit être installé sur la plate-forme"*. Pouvoir attester de la conformité d'une plate-forme élévatrice au regard de cette exigence, nécessite de disposer de la norme "EN ISO 13850".

Normalisation syntaxique

La normalisation syntaxique a pour but la réécriture en une règle simple, atomique en employant un phrasé plus standard - c'est-à-dire ***si ... alors*** - ou alors la défragmentation d'une règle complexe en un ensemble de règles simples. Dans la littérature, par exemple [Siddharthan 2006], il existe plusieurs sous-types de normalisations syntaxiques. Comme dans le travail Guissé *et al.* [2012], nous ne gardons que celles appropriées pour notre contexte, c'est-à-dire dédiées aux règles.

— La ***réorganisation des phrases*** (*sentence reordering*) qui vise à réorganiser les termes d'une exigence de sorte que le *logical rule pattern* (si ... alors) soit explicite.
Exemple :

- **Règle originale :** *"La vitesse de la plate-forme ne doit pas dépasser 0,15 m/s."*
- **Règle après réorganisation : Si** *la plate-forme a une vitesse supérieure à 0.15m/s* **alors** *elle présente un défaut.*

— L'***explicitation des énumérations*** (*splitting of enumerations*) est un écueil majeur pour toute application dans le domaine du traitement automatique du langage naturel. Les énumérations peuvent être signalées par : la présence de ***conjonctions de coordination*** (***et***, ***ou***), l'emploi d'une ***paire de connecteurs*** (exemple : ***ni . . . ni***) ou la ***juxtaposition***. Reformuler une prescription comportant des énumérations, revient à la décomposer en des prescriptions atomiques.
Exemple :
- **Règle originale :** *"La largeur de passage libre de la plate-forme, de ses accès et des accès palier ne doit pas être inférieure à 800 mm."*
- **Règles après explicitation des énumérations :** (**1**) *La largeur de passage libre de la plate-forme ne doit pas être inférieure à 800 mm.* (**2**) *La largeur des accès à la plate-forme ne doit pas être inférieure à 800 mm.* (**3**) *La largeur des accès palier ne doit pas être inférieure à 800 mm.*

Notons que si les marqueurs d'énumérations sont explicites dans bien des cas, il est parfois très difficile d'exhiber les items d'une énumération et par la même occasion d'identifier les éléments, de la phrase, communs aux différents items.

La restauration sémantique

Habituellement, pour résoudre une co-référence, comme dans le cadre de la décontextualisation présentée plus haut, on se sert des termes présents autour de la référence implicite. Dans certains cas, la résolution des co-références fait impérativement appel à de nouveaux termes, non présents dans le contexte de la référence implicite. Dans ces cas, on parle de ***restauration sémantique***. Exemple :

- **Règle originale :** *"Un engrènement correct signifie que le diamètre du cercle primitif du pignon doit coïncider avec au maximum 1/3 du module au-delà du cercle primitif de la crémaillère."*
- **règle après restauration** (**1**) **Le chevauchement pignon-crémaillère** *est le rapport entre le diamètre du cercle primitif du pignon et le module au-dela du cercle primitif de la crémaillère.* (**2**) **Le chevauchement pignon-crémaillère** *doit être inférieur à 1/3.*

L'élagage

Cette dernière catégorie de normalisation n'est pas mentionnée dans le travail de Guissé *et al.* [2012] et est un ajout proposé dans le cadre de cette thèse. L'***élagage*** (*pruning*) consiste à supprimer d'une phrase réglementaire une proposition (au sens de la grammaire [4]) dont ***la contribution à la formalisation*** (dans le cadre tu contrôle réglementaire

4. Structure élémentaire d'une phrase, formée autour d'un verbe. Équivalent de l'anglais *clause*.

tout au moins) de ladite phrase est négligeable ou nulle [5]. C'est le cas des propositions expliquant ou justifiant le pourquoi d'une contrainte donnée. Exemple :

— **Règle originale :** *"Pour permettre aux usagers d'entrer et de sortir librement de la plate-forme élévatrice, la temporisation de la porte doit être initialement de 5 s".*
— **Règle après élagage :** *"La temporisation de la porte doit être initialement de 5 s".*
Pour formaliser la règle originale, la proposition ***"Pour permettre aux usagers d'entrer et de sortir librement de la plate-forme élévatrice"*** n'est pas pertinente. Elle sert uniquement à expliquer au lecteur, le choix de la valeur initiale de la temporisation de la plate-forme.

Au cours de notre travail de thèse, nous nous consacrons aux règles ne devant nécessiter d'autres transformations que les normalisations lexicales, syntaxiques ainsi que l'élagage. Les exigences nécessitant une résolution de co-référence ou une restauration sémantique sont hors de la portée de notre approche de formalisation automatique des règles.

3.1.4 Règles formalisables et non formalisables

Lorsqu'un expert métier envisage de fournir une version formelle d'une règle, sur la base d'un vocabulaire contrôlé, il peut se retrouver confronter à quatre cas de figure. Ces cas sont répertoriés par Yurchyshyna [2009] dans ces travaux de thèse.

Cas 1 La règle reprend exactement les termes tels que mentionnés dans le vocabulaire contrôlé (c'est-à-dire pas de normalisation lexicale nécessaire - voire page 25). Ces règles sont qualifiées de ***complètement interprétables*** (*completely interpretable rules*).

Cas 2 La règle requiert une normalisation lexicale pour devenir complètement interprétable. Dans ce cas elle est appelé ***règle reformulée*** (*reformulated rule*).

Cas 3 Le vocabulaire contrôlé ne dispose de tous les termes nécessaires pour exprimer de manière formelle l'exigence réglementaire. Ce type de règle est qualifiée de ***partiellement interprétables*** (*partially interpretable rules*).

Cas 4 Il se peut qu'une ne puisse en aucun cas faire l'objet d'une réécriture formelle dans les termes du vocabulaire contrôlé, fût-il enrichi. Exemples : (1) ***"La descente contrôlée doit être possible en toutes circonstances."*** (2) ***"Des mesures doivent être prises pour empêcher la pénétration de corps étrangers entre chaque pignon d'entraînement ou de sécurité et la crémaillère."***

Lorsque nous regardons de plus près la classification ci-dessus, proposée par Yurchyshyna [2009], nous pouvons conclure que : si le vocabulaire contrôlé est supposé complet (aucun enrichissement nécessaire), alors on peut diviser les règles en deux catégories à savoir ***formalisables*** ou ***interprétables*** (cas 1-3) et ***non-formalisables*** ou ***non-***

5. Jugement de valeur émis par les experts métiers.

interprétables. Cette classification simplifiée a, par exemple, été adoptée au cours des travaux de Bouzidi [2013]. Dans le cadre de cette thèse, notre attention portera uniquement sur les règles formalisables.

3.1.5 Synthèse : Quels types de règles pour notre méthode de formalisation ?

Tout au long de la section 3.1, nous avons présenté les différentes manières que les chercheurs ont d'appréhender les règles et les prescriptions réglementaires. Ce regard sur les règles fait suite à celui exposé, au chapitre précédent, à la section 2.1. Nous avons ainsi pu, pas-à-pas, délimiter le périmètre des règles pour lesquelles nous proposons une méthode automatique de formalisation. En résumé :

1. Seules les exigences ***textuelles*** seront considérées dans notre approche de formalisation automatique de formalisation. De plus, tous les termes nécessaires à la compréhension de ces exigences devront être contenues dans une seule et même ***phrase*** (cf. section 2.1.1, page 10).
2. Les prescriptions concernées par notre travail devront être ***strictes***, c'est-à-dire des ***obligations***. Notons que les ***interdictions*** sont considérées comme des obligations [6], contrairement aux ***permissions*** et aux ***recommandations*** (cf. section 3.1.1). Ces dernières ne permettent pas d'affirmer si oui ou non, un objet présente une anomalie.
3. Le travail effectué dans le cadre des spécifications de RuleML nous a permis de mettre en lumière deux grandes familles de règles : les *reaction rules* (qui exigent une action) et les *deliberation rules* dont font partie les *derivation rules*(qui sont de simples inférences) - cf. section 3.1.2. Seules les *derivation rules*, utilisables pour un contrôle automatique de conformité, sont à la portée de notre méthode de formalisation des exigences.
4. Les travaux de thèse de Guissé [2013] ont mis en lumière un ensemble de challenges à surmonter lorsque se pose la question de la formalisation des règles. A ces challenges, Guissé *et al.* ont donné le nom de normalisations (voir section 3.1.3). À ces normalisations, nous avons trouvé pertinent d'ajouter celui de l'***élagage*** (page 27). Seules les exigences ne requérant pas de normalisation(s), ou nécessitant une ***normalisation lexicale*** (page 25) et/ou ***syntaxique*** (page 26) et/ou un ***élagage*** seront considérées dans notre approche de formalisation.
5. Seules les prescriptions ***complètement interprétables*** (cf. section 3.1.4), au regard d'un vocabulaire contrôlé défini, seront susceptibles d'être formalisées automatiquement par notre approche.

Nous venons de présenté les types de règles ainsi que les règles qui seront considérées dans notre approche de formalisation automatique de règles. La suite de notre revue de littérature concerne les langages qui permettent de représenter les règles de manière

6. Une interdiction est une obligation de faire le contraire.

formelle.

À présent, nous nous intéressons aux langages de représentations pour les données que nous manipulons ainsi que pour les règles formelles.

3.2 Langages de représentation des connaissances

Il existe une pléthore de langages pour la représentation formelle des connaissances. En fonction de l'***utilisation*** qu'on veut faire des informations formalisées, de l'***expressivité*** que l'on souhaite, de la manière dont les données seront exposées ou échangées, etc., on peut choisir de recourir à tel langage ou plutôt à tel autre. Nous n'avons pas l'ambition de présenter une liste des "principaux" langages ou familles de langages qui existent et sont utilisés en ingénierie des connaissances. Dans cette section, nous nous intéresserons uniquement aux principaux langages de représentation des connaissances utilisés dans la communauté du ***Web Sémantique***.

Web Sémantique

Le Web Sémantique (*Semantic Web*) encore appelé ***Web des données*** (*Web of data*) est "une extension du Web actuel dans lequel l'information est annotée de façon à être claire et non-ambiguë, favorisant ainsi les échanges entre les ordinateurs et leurs utilisateurs."[a] [Berners-Lee *et al.* 2001]. Plus loin dans le même article, pour nous faire mieux entrevoir les possibilités apportées par cette extension du Web, Berners-Lee, Hendler, & Lassila soulignent que le Web Sémantique permet aux ordinateurs de traiter directement ou indirectement les données et aussi de déduire de nouvelles connaissances.

a. Traduit de l'anglais.

Ce choix est motivé par les caractéristiques que présentes les standards du Web Sémantique. Parmi ces caractéristiques, on retrouve :

— L'***interopérabilité***. Lorsque nous avons présenté la gestion des données du bâtiment au travers du BIM, nous évoquions déjà (section 2.2.1) la nécessité de tenir compte de l'interopérabilité pour le bon partage des données. Les langages du Web Sémantique disposent d'axiomes qui permettent de modéliser des entités de différentes manières, selon les points de vue, les contextes, les cas d'utilisations [Heflin 2004].

— Le ***partage de données***. C'est une notion qui va de pair avec l'interopérabilité. Les résultats issus d'un travail de conceptualisation ou de modélisation dans le monde du Web Sémantique, ont généralement vocation à être diffusés. Ceci permet la réutilisation, l'adaptation et l'extension de ces données, dont, rappelons-le, la sémantique est claire. Dans ce cas aussi, les standards du Web Sémantique répondent à ces besoins [Heflin 2004].

— Les ***mécanismes d'inférence***. Dans le contexte du Web Sémantique, les inférences sont synonymes de déduction de faits nouveaux [W3C 2009]. Pour le Web Sémantique, les inférences permettent de s'assurer de la qualité de l'intégration des données du Web par la découverte de nouvelles informations et aussi par le fait de s'assurer que la coexistence des informations anciennes et de celles nouvellement déduites est ***cohérente***. Autrement dit qu'il n'existe pas de ***contradiction***. Il est important de noter que, pour certaines données, le non-usage d'informations nouvelles obtenues à partir d'inférences sur ces données, compromet l'utilisation de ces données. Pour illustration, supposons une base de connaissances dans laquelle nous avons les trois (03) informations (*i*) "**Porte-1** est une sorte de **porte de secours**", (*ii*) "**Porte-2** est une sorte de **porte**" et (*iii*) "Toute **porte de secours** est aussi une **porte**". Supposons de plus une exigence réglementaire qui requiert que toutes les **portes** aient une largeur donnée. Sans effectuer d'inférences, un contrôle de conformité sur cette base de connaissances s'attardera uniquement sur la vérification de la largeur de **Porte-2**; car c'est la seule entité explicitement déclarée comme étant une **porte**. Or, en raisonnant, on déduira des faits (*i*) et (*iii*) la nouvelle information (*iv*) "**Porte-1** est une sorte de **porte**". Ainsi, le contrôle de conformité portera à la fois sur **Porte-1** et **Porte-2**. Nous voyons que la non-déduction d'une information pourrait conduire à un contrôle de conformité partiel, voire faux.
— Un ***équilibre entre expressivité et scalabilité***. De milliards de pages composent déjà le Web et l'application du Web Sémantique, qui ajoute et génère de nouvelles informations, doit pouvoir se faire avec un temps de réponse adéquat. Par ailleurs, l'efficacité des applications du Web Sémantique ne doit pas nécessairement sacrifier l'expressivité. Heflin [2004] nous dit que l'expressivité détermine ce qui peut être dit dans un langage et par conséquent conditionne le potentiel des inférences qui peuvent être faites à l'aide de ce langage. Plus un langage est expressif, plus il dispose de primitives permettant de représenter une grande diversité d'informations. Une fois de plus, dans la famille des langages du Web Sémantique, plusieurs langages existent et permettent, en fonction des applications, de trouver un compromis entre expressivité et scalabilité.
— La ***standardisation***. En plus des qualités citées précédemment, les langages du Web Sémantique ont l'avantage d'être des standards, reconnus et utilisés par une large communauté.

Par ailleurs, soulignons qu'en faisant ce choix, nous assurons la continuité et la réutilisation des travaux de thèses précédents ([Yurchyshyna 2009], [Bouzidi 2013]) dans le domaine du contrôle automatique de la conformité réglementaire.

3.2.1 Principaux langages du Web Sémantique

De nombreux standards et langages composent la famille du Web Sémantique. La désormais populaire *Semantic Web Stack*[7], reprise en figure 3.2, permet d'avoir une vue pertinente de l'organisation et de la relation entre ces langages. Chaque couche de cette pile exploite et étend les possibilités des couches inférieures.

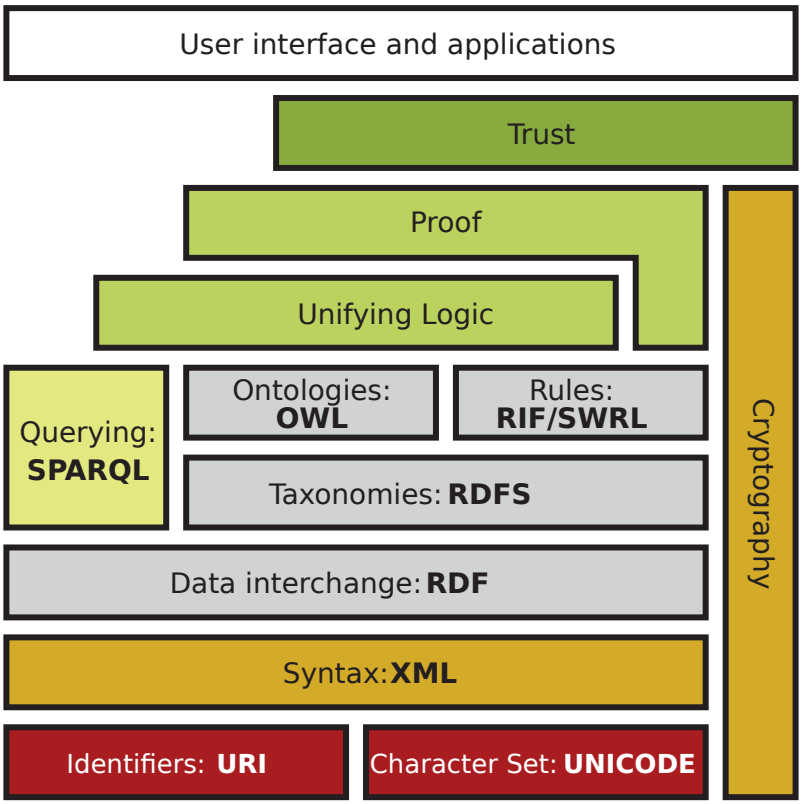


FIGURE 3.2 – *Semantic Web Stack* (source `https://commons.wikimedia.org/wiki/File:Semantic_web_stack.svg`)

Dans les paragraphes qui suivent, nous présentons les principaux langages que l'on trouve au niveau des couches en relation avec ce travail de thèse.

Ressource Description Framework (RDF)

RDF est la première brique des standards du Web Sémantique et recouvre à la fois un modèle et plusieurs syntaxes pour publier des données à propos de tout élément sur le web [Cyganiak *et al.* 2014]. RDF est un cadre de description de ressources dans le sens où [Gandon *et al.* 2012] :

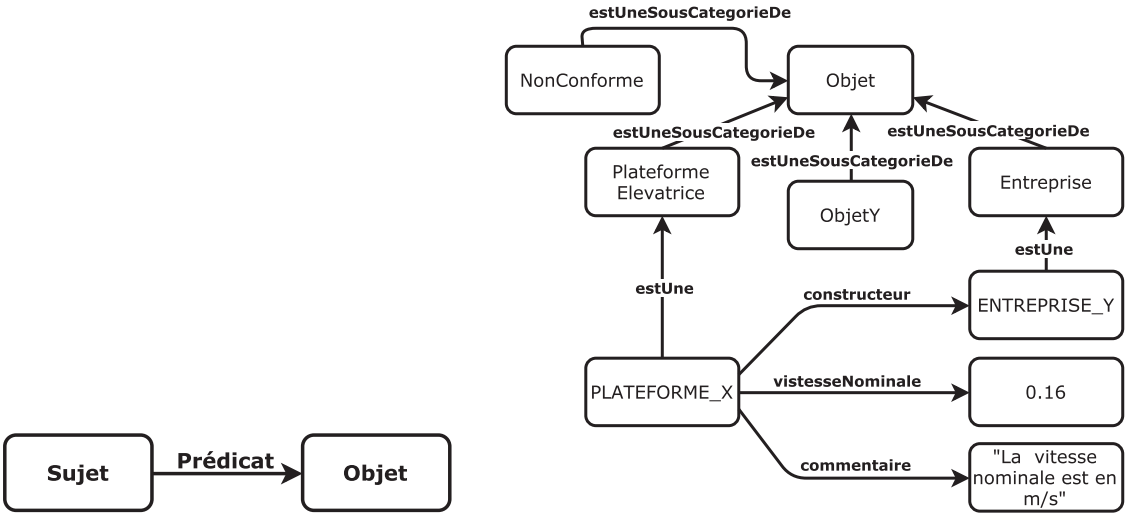
— Les ***ressources*** sont à la base du Web Sémantique : tout ce à quoi on peut se référer est une ressource (exemple : une page web, une image, un lieu, une personne, un service, un produit, etc.).

7. Aussi connue sous les noms de *Semantic Web Cake*, *Semantic Web Cake Layer* et *Semantic Web Pyramid*

— La ***description*** d'une ressource est un ensemble d'attributs, de caractéristiques, de propriétés et de relations notamment avec d'autres ressources.
— Le cadre standardise les modèles, les langages et syntaxes des descriptions.

En bref, RDF fournit une structure de données standard et un modèle pour encoder des données sur n'importe quelle entité sur le web ; ces entités sont des ressources et sont identifiées de manière unique à l'aide d'*Uniform Resource Indentifiers*[8] (URIs).

Au coeur de la représentation des connaissances en RDF, se trouve des atomes de ***triplets***. Un triplet RDF pemet de décrire une ressource en associant celle-ci à une propriété et à la valeur de cette propriété. C'est un ***énoncé*** (en anglais *statement*) calqué sur le modèle d'une ***phrase simple***[9] c'est-à-dire formé selon le schéma ⟨ ***sujet + verbe/prédicat + objet*** ⟩. Ici la ressource correspond au sujet, la propriété au prédicat et la valeur de la propriété fait office d'objet.



(a) Le triplet RDF (b) Exemple simplifié de données RDF

FIGURE 3.3 – Exemple de représentation des connaissances en RDF.

La figure 3.2 illustre la notion de triplet RDF (figure 3.3a) et donne un exemple de triplets RDF formant une sorte de graphe (figure 3.3b). Sur cette figure, on peut observer :

— Un triplet RDF est définit par deux ***noeuds***, reliés par un ***arc orienté***, du sujet vers l'objet. Cet arc est étiqueté par un ***label*** qui n'est rien d'autre que le ***prédicat*** du triplet.
— Le sujet d'un triplet est nécessairement une ressource, tandis que l'objet peut être une ressource ou un ***littéral***.

8. On retrouve parfois la terminologie *Internationalized Resource Identifiers* (IRIs) en lieu et place d'URI. Les adresses IRIs généralisent et internationalisent les URIs, notamment par l'emploi des caractères de nombreux alphabets (arabes, latins, asiatiques, etc.) `http://tools.ietf.org/html/rfc3987`
9. Dans les langues telles que l'Anglais, le Français, l'Allemand, etc.

Littéral
"Un ***littéral*** est une chaîne de caractères arbitraires typée ou non typée représentant des valeurs telles que du texte, des entiers, des dates, etc." [Gandon *et al.* 2012].

Par son modèle qui oblige à représenter un fait, non atomique, ou un ensemble de faits à l'aide de triplets, RDF permet d'encoder une très grande variété de connaissances.
Toutefois, le modèle de données RDF démeure un modèle abstrait qu'il est nécessaire de doter d'une syntaxe formelle, compréhensible par un ordinateur. Il existe plusieurs formats de sérialisations de graphes RDF et, parmi les plus répandus, on trouve : *RDF/XML*, *N-Triples*, le *Terse RDF Triple Language* (*Turtle*) et *N3*. Le *listing* 3.1 fournit un exemple d'ontologie suivant la syntaxe RDF/XML. Ce même exemple est représenté, dans le *listing* 3.2, suivant la syntaxe Turtle.
Les exemples de données RDF que nous fournirons dans le reste de ce manuscrit, utiliserons la syntaxe Turtle ; ceci à cause de sa compacité et de sa lisibilité.

Listing 3.1 – Exemple d'ontologie en syntaxe RDF/XML

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <owl:Ontology rdf:about="http://example.com/LiftingPlatformOntology#">
  </owl:Ontology>

  <owl:Class rdf:about="http://example.com/LiftingPlatformOntology#LiftingPlatform">
    <rdfs:label xml:lang="en">lifting platform</rdfs:label>
    <rdfs:label xml:lang="fr">plateforme élevatrice</rdfs:label>
  </owl:Class>

  <owl:DatatypeProperty rdf:about="http://example.com/LiftingPlatformOntology#ratedSpeed">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
    <rdfs:domain rdf:resource="http://example.com/LiftingPlatformOntology#LiftingPlatform"/>
    <rdfs:label xml:lang="en">rated speed</rdfs:label>
    <rdfs:label xml:lang="fr">vitesse nominale</rdfs:label>
  </owl:DatatypeProperty>

</rdf:RDF>
```

Listing 3.2 – Exemple d'ontologie en syntaxe Turtle (reprend l'ontologie du listing 3.1)

```
@prefix : <http://example.com/LiftingPlatformOntology#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

```
<http://example.com/LiftingPlatformOntology#> rdf:type owl:Ontology .

:LiftingPlatform
        rdf:type owl:Class;
        rdfs:label "lifting platform"@en,
        "plateforme élevatrice"@fr .

:ratedSpeed
        rdf:type owl:DatatypeProperty;
        rdfs:range xsd:double;
        rdfs:domain :LiftingPlatform;
        rdfs:label "rated speed"@en,
        "vitesse nominale"@fr .
```

Afin de fluidifier la présentation des langages qui sous-tendent les autres niveaux de la *Semantic Web Stack*, il est nécessaire pour nous de donner une définition du terme ***ontologie*** ; terme qui désigne une notion essentielle dans notre travail.

Ontologie

L'ontologie est la structure de données, au coeur des modèles du Web Sémantique. C'est un terme emprunté au domaine de la philosophie dans lequel il est défini en tant que branche de la connaissance qui étudie les propriétés générales de ce qui existe [Gandon *et al.* 2012]. En informatique, une ontologie est un vocabulaire partagé qui conceptualise un domaine. Elle est généralement composée d'une hiérarchie de concepts pertinents dans ce domaine (aussi appelée ***taxonomie***) , de leurs propriétés, des relations entre ces entités, ainsi que des règles et des axiomes qui les contraignent [Gandon *et al.* 2012], [Arvidsson & Flycht-Eriksson 2008]. De ce fait, une ontologie fournit un référentiel pour construire un système de base de connaissances et permettre les inférences pour la recherche d'informations, la gestion des connaissances, etc. [Gandon *et al.* 2012].

Pour des besoins de lisibilité et de simplification de notation dans nos exemples, nous adoptons les abréviations des espaces de nommage fournies dans le tableau 3.1.

Tableau 3.1 – Principaux espaces de nommage utilisés dans ce document et leur préfixe.

Préfixe	**URI de l'espace de nommage**
`rdf`	http ://www.w3.org/1999/02/22-rdf-syntax-ns#
`rdfs`	http ://www.w3.org/2000/01/rdf-schema#
`owl`	http ://www.w3.org/2002/07/owl#
`xsd`	http ://www.w3.org/2001/XMLSchema#
(vide)	http ://example.com/LiftOnto#

Ontologies légères : RDF Schema

RDF Schema, abrégé RDFS, est le langage de données de description de données historiquement rattaché à RDF. RDFS permet de spécifier les ontologies dites ***légères***, c'est-à-dire de nommer les classes (c'est-à-dire les concepts), les propriétés, de donner les classes que l'on peut relier par ces propriétés, et de définir une organisation hiérarchique pour ces classes et ces propriétés [Brcikley & Guha 2014]. De manière formelle, RDFS s'appuie sur les axiomes suivants - les exemples s'appuient sur la modélisation de la figure 3.3b :

— `rdfs:Class` qui permet de déclarer qu'une ressource est une classe. `:PlateformeElevatrice` et `:Entreprise` sont des exemples de `rdf:Class`.
— `rdf:Property` est la classe (autrement dit c'est une instance de `rdfs:Class`) qui regroupe les propriétés RDF. Comme exemples de `rdf:Property`, nous avons `:constructeur` et `:vitesseNominale`.
— `rdfs:subClassOf` est une propriété qui permet d'affirmer qu'une classe est une sous-catégorie d'une autre classe. C'est cette propriété qui permet de construire les relations hiérarchiques, encore appelées ***subsomptions***. Un exemple de subsomption est la relation hiérarchique entre classes `:Entreprise` et `:Objet`.
— `rdfs:Litteral` est la classe qui regroupent les littéraux. Ainsi les valeurs 0.16 et "la vitesse nominale est en m/s" sont des instances de cette classe.
— `rdf:type` est une instance de `rdf:Property` utilisée pour énoncer qu'une ressource est une instance d'une classe.
— `rdfs:domain` est une instance de `rdf:Property` qui permet de déclarer les classes dans lesquelles une propriété prend ses sujets. Par exemple, on pourrait dire `:vistesseNominale rdfs:domain :PlateformeElevatrice`.
— `rdfs:range` est une instance de `rdf:Property` qui permet de déclarer les classes des valeurs possibles d'une propriété. Exemple : `:constructeur rdfs:range :Entreprise`.

Ontologies lourdes : OWL

Le langage OWL (*Web Ontology Language*) est ***le*** langage standard pour la définition des ontologies pour le web des données. Il est plus expressif que RDFS et permet, en plus des relations hiérarchiques des classes et des propriétés, d'exprimer des notions d'équivalence de classes ou de propriétés, d'égalité de ressources, de différence, de contraire, de symétrie, de cardinalité, etc.

OWL classe les propriétés dans 3 catégories, les *datatype properties*, *object properties* et les *annotation properties* (qui permettent d'ajouter des éléments descriptifs tels que des étiquettes, des commentaires, des informations de versions, etc., aux entités).

Datatype property
Une *datatype property* est une propriété OWL dont le *range* (`rdfs:range`) est un littéral. Exemple :

```
:vitesseNominale rdf:type owl:DatatypeProperty;
                 rdfs:domain :PlateformeElevatrice;
                 rdfs:range xsd:float.
```

Object property
Une *object property* est une propriété OWL dont le *range* (`rdfs:range`) est une classe. Exemple :

```
:constructeur rdf:type owl:ObjectProperty;
              rdfs:domain :Objet;
              rdfs:range :Entreprise.
```

En OWL, il existe différentes manières de décrire une classe. Ainsi, une classe peut être le ***complément***, la ***sous-classe***, l'***équivalent***, l'***union***, l'***intersection*** d'une ou plusieurs classes, ou même être décrite par la ***liste exhaustive de ses instances*** ou par des ***restrictions***.
En plus des classes et des propriétés, on trouve dans les ontologies OWL un troisième type d'entité : les ***individus*** (parfois appelés ***instances***).

Individu
En RDF, un ***individu*** est représenté par une URI (on parle aussi d'***individu nommé***) ou une ressource anonyme (c'est-à-dire sans URI) et pouvant être typé avec des classes et caractérisé à l'aide de propriétés [Bock *et al.* 2012]. En OWL, il est nécessaire que ces classes et propriétés utilisées dans la description de l'individu soient au préalable définies dans l'ontologie. Une description d'un individu est appelée une ***assertion*** [Gandon *et al.* 2012]. En OWL, tout individu est une instance de la super classe `owl:Thing`. Comme exemples d'individus sur la figure 3.3, nous avons `:PLATEFORME_X` et `:Entreprise_Y` et on écrit

```
:PLATEFORME_X rdf:type :PlateformeElevatrice.
:ENTREPRISE_Y rdf:type :Entreprise.
```

Bien que nous employions l'expression "OWL" pour parler d'un langage, cette même expression peut aussi désigner une famille de langages. Cette famille de langages hérite des logiques de descriptions [10], qui elles-mêmes sont une famille de fragments de la logique du premier ordre [11], chaque fragment ayant une sémantique formelle et des propriétés computationnelles connues. Chaque membre de la famille OWL permet de

10. http ://dl.kr.org/
11. https ://fr.wikipedia.org/wiki/Calcul_des_prédicats

préciser la position dans laquelle on se situe dans un espace bidimensionnel dont les axes sont l'***expressivité*** et la ***scalabilité*** (cf. section 3.2).
Dans la première version de OWL, on trouve trois (03) familles de langages (voir figure 3.4a) dont l'expressivité est décroissante. Du plus expressif au moins expressif, nous avons [Gandon *et al.* 2012] :

— ***OWL Full*** permet de mélanger librement les déclarations de OWL 1 avec celles de RDFS, mais la ***décidabilité*** [12] des algorithmes n'est pas assurée.
— ***OWL DL*** [13] restreint OWL Full et a été conçu pour conserver le plus gros fragment décidable de OWL Full en s'assurant en plus de la ***complétude*** [14].
— ***OWL Lite*** est un langage à l'expressivité réduite qui conserve une fraction de OWL DL afin de s'alléger la charge de calcul.

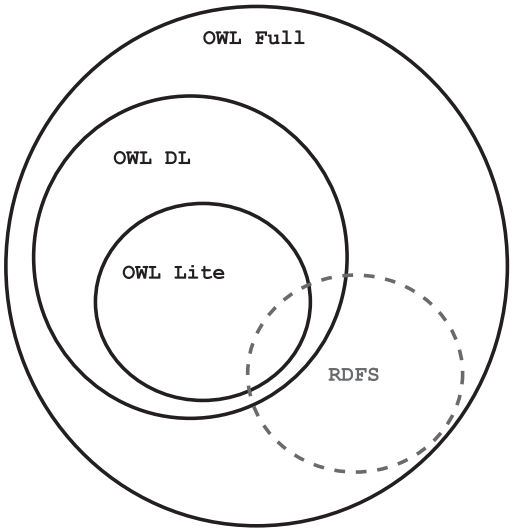
Avec OWL 2, on parle plutôt de ***profils*** (voir figure 3.4b). Ces profils sont issus du fait qu'en pratique, les ontologies n'ont ni les mêmes caractéristiques (nombre d'entités, expressions de classes plus ou moins complexes, etc.), ni les mêmes exigences (expressivité vs scalabilité, compatibilité avec des bases de données, interopérabilté avec des moteurs de règles, etc.) [Patel-Schneider 2012] :

— ***OWL 2 EL*** (*Existential Language*) renferme l'expressivité nécessaire pour les ontologies de très grandes tailles (par exemple *SNOMED CT* [15] et le *NCI Thesaurus* [16] qui ont une taille de l'ordre de 10^5).
— ***OWL 2 QL*** (*Query Language*) est orienté ontologie légère et a une expressivité proche de celle fournie dans les bases de données traditionnelles.
— ***OWL 2 RL*** (*Rule Language*) est un profil recommandé pour les applications nécessitant un passage à l'échelle des raisonnement, qui ont un temps polynomial par rapport à la taille de l'ontologie.

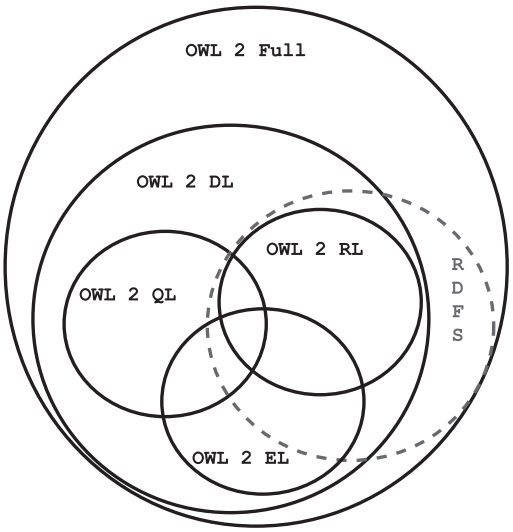
IfcOWL : Renforcer les IFC par le langage OWL

IFC (Industry Foundation Classes) est le standard de représentation et de partage de données dans le domaine de la Construction (voir section 2.2). Malheureusement, les IFC sont limités lorsqu'il s'agit de répondre à certains besoins. Par exemple, le standard IFC montre ses limites pour des besoins de réutilisation des connaissances d'autres domaines, pour le contrôle de conformité, etc. [Pauwels *et al.* 2011b].
Cela vient du fait que les IFC, représentés dans le langage EXPRESS, n'ont pas été conçus pour interagir avec les vocabulaires et autres bases de connaissances existantes. Ainsi, lorsqu'on travaille avec les IFC, on ne peut travailler qu'avec des données déjà exprimées en IFC. Il est impossible, de s'appuyer par exemple, sur une base de connaissances sur

12. Un algorithme est dit décidable si on a l'assurance que dans tous les scénarios, il se termine en un temps fini.
13. DL est mis pour *Description Logic* (logique de description).
14. Un algorithme d'inférence est dit complet s'il permet d'inférer toutes les conclusions possibles à partir d'une base de connaissances.
15. Terminologie du domaine de la santé. http ://www.ihtsdo.org/snomed-ct
16. Ontologie biomédicale. https ://ncit.nci.nih.gov/ncitbrowser/



(a) Famille de langages de OWL 1. **DL** = Description Logic.



(b) Profiles de OWL 2 . **RL** = Rule Language. **QL** = Query Language. **EL** = Existential Language.

FIGURE 3.4 – Organisation des langages dans OWL 1 et OWL 2 (source [Hoekstra 2009]

les matériaux pour caractériser un matériau dans un schéma IFC. De même, si l'on veut référencer des données géographiques d'une installation en IFC en s'appuyant sur une source de données géographiques non IFC, cela ne peut pas être fait en EXPRESS. Par ailleurs, si un utilisateur est intéressé par une partie du schéma IFC, par exemple une entitié ou une énumération, il ou elle ne peut pas uniquement référencer cette entité ou cette énumération. De plus, les IFC se prêtent mal au besoin, plus que jamais urgent, du contrôle automatique de la conformité réglementaire. En effet, le langage EXPRESS dans lequel est exprimé les IFC, ne supporte pas de manière native les mécanismes d'inférences complexes que requiert le contrôle de conformité.

Pour chacun des inconvénients des IFC mentionnés ci-dessus, les outils et technologies du Web Sémantique apportent des réponses pertinentes, comme nous l'avons illustré à l'introduction de la section 3.2. Aussi, pour pouvoir tirer pleinement profit des forces des paradigmes du Web Sémantique, il est nécessaire de disposer d'une nouvelle représentation des IFC s'appuyant sur le standard RDF. Plusieurs initiatives ont vu le jour à ce sujet, avec pour objectif de transformer le schéma original IFC, exprimé en EXPRESS, dans un format RDF/OWL [Hoang & Törmä 2014], [Barbau *et al.* 2012], [Krima *et al.* 2009], [Pauwels & Terkaj 2016], [de Farias *et al.* 2015]. Les ontologies IFC résultant de ces initiatives offrent la possibilité de réutiliser tout ou partie des IFC, d'étendre les IFC (ajout de nouvelles entités) (cf. section 6.1), profiter des mécanismes d'inférences du Web Sémantique (cf. chapitres 4 et 5). Un extrait de l'ontologie IfcOWL [Pauwels & Terkaj 2016] est présenté à la figure 3.5.

En regardant la légende de la figure 3.5, on peut remarquer que :

1. Nous avons pris le soin de distinguer les classes représentant des *IFC relationship entities* des autres classes. Ces classes, qui sont des sous classes de `IfcRelationship`,

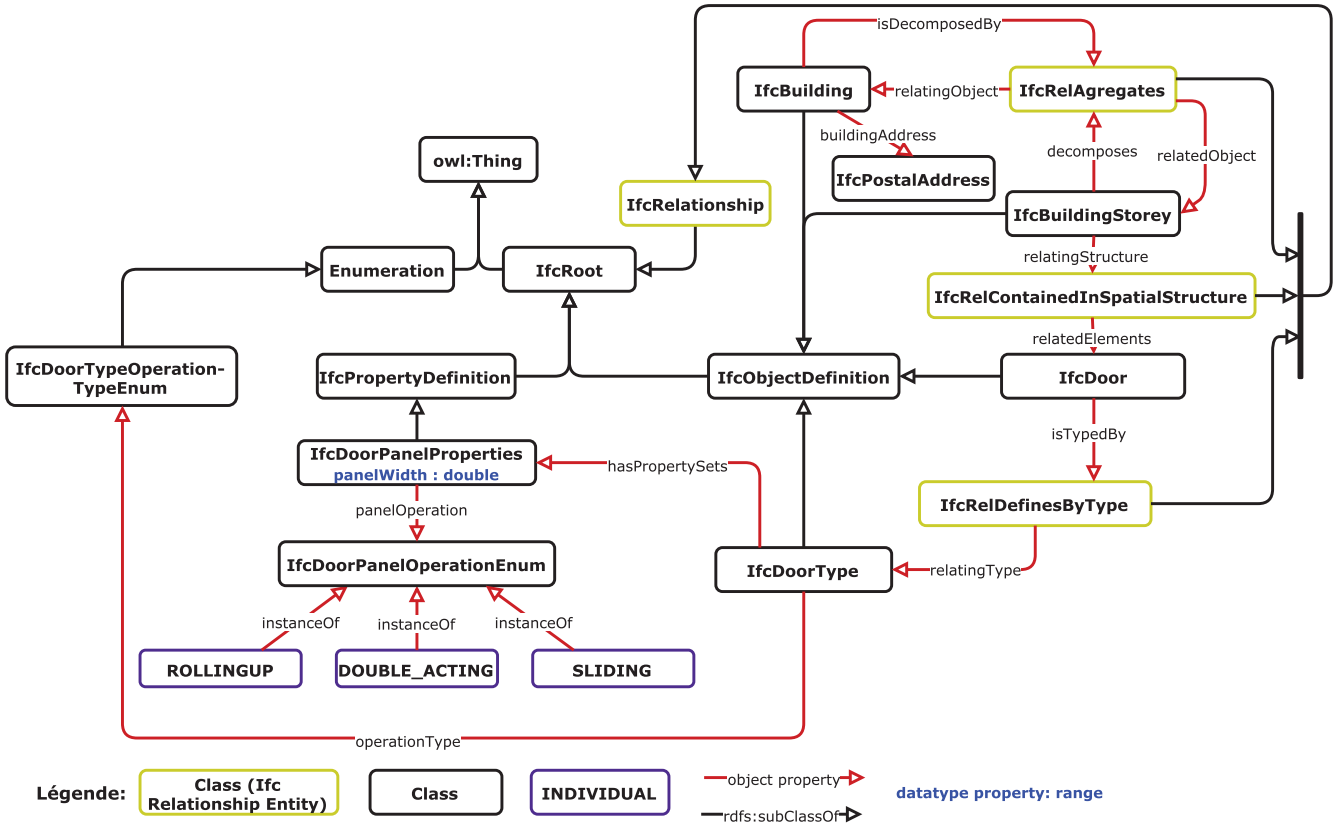


FIGURE 3.5 – Extrait de l'ontologie IfcOWL [Pauwels & Terkaj 2016].

matérialisent les relations entre les ***objets IFC***. Nous entendons par objets IFC, les instances de la classe `IfcObjectDefinition`. C'est la super-classe de tous les physiques tangibles (murs, poutres, revêtements, etc.), ou ayant une existence physique, comme les espaces, ou encore des objets conceptuels (par exemples les maillages ou les frontières) [Liebich *et al.* 2013].

Pour relier, de manière formelle, deux objets en IFC, il est nécessaire de faire appel à une ***relation objectifiée***. Pour illustration, prenons la prescription "*Each building must have a storey fitted with fire doors*" [17]. Dans cette exigence en langage naturel, on peut observer que les entités "*building*" et "*storey*" sont directement reliées par un verbe ("*have*"). Or, en regardant le schéma IFC à la figure 3.5, pour relier les entités `:IfcBuilding` et `:IfcBuildingStorey`, il est nécessaire : (*i*) via un premier predicat, `:isDecomposedBy`, d'obtenir la liste des objets (`:IfcRelAgregates`) qui composent le bâtiment(`:IfcBuilding`) (*ii*) via un second prédicat, `:relatedObject`, à partir de `:IfcRelAgregates`, d'obtenir la liste des étages (`:IfcBuildingStorey`) du bâtiment. Une remarque similaire peut être faite entre la connexion des entités, en langage naturel, "*storey*" et "*fire doors*" en utilisant l'unique verbe "*fitted*". En effet, pour se plier au formalisme des IFC, il est nécessaire de décomposer l'étage d'un bâtiment (`:IfcBuildingStorey`) en des éléments contenus à cet étage (c'est-à-dire des ins-

17. "Tout bâtiment doit posséder au moins un étage muni de portes coupe feu."

tances de `:IfcRelContainedInSpatialStructure`); et c'est parmi ces objets qu'on identifiera les portes (`:IfcDoor`) et donc les portes coupe-feu [18].

2. Les libellés des termes IFC sont construits sur la langue anglaise. Certains de ces libellés ne sont pas toujours bâtis sur des mots du dictionnaire anglais. Autrement dit, ***ils ne sont pas naturels*** et par conséquent on ne les retrouve pas, tels quels, dans des documents normatifs. Ce caractère artificiel des labels des entités IFC, se remarque :
   — Par l'ajout du suffixe `Ifc` au début de certains libellés. Par exemple `IfcDoor`, `IfcBuilding`, `IfcBuindingStorey`, etc.
   — Par la présence d'abréviations dans les noms de plusieurs entités. C'est le cas de `Ifc`**`Rel`**`Agregates`, `Ifc`**`Rel`**`DefinesByType`, `IfcDoorTypeOperationType`**`Enum`**, etc.

Les deux (2) points relevés ci-dessus nous permettent d'affirmer que les libellés des entités IFC présentent une ***naturalité faible***. Nous entendons par naturalité d'un langage, son rapport à la lisibilité et la compréhensibilité du langage naturel [Kuhn 2014]. Nous reprenons cette notion, telle que présentée par Kuhn, avec plus de détails à la section 3.4.3 sur les langages naturels contrôlés.

3.2.2 Langages de règles pour le Web

Nous entendons par ***langages de règles pour le Web*** (*Web rule languages*) des langages de règles basés sur les standards du Web. Ils fournissent l'expressivité nécessaire pour rendre possible l'interprétation des données par les ordinateurs et la traduction vers d'autres langages de règles. Les langages de règles sont aussi utilisés par les moteurs d'inférence et servent parfois à assurer l'interopérabilité et l'échange de règles sur le Web.

> **Règle d'inférence**
> Une règle d'inférence est une fonction qui prend en entrée un n-uplet de formules et retourne une nouvelle formule. Les entrées sont appelées ***prémisses*** ou ***antécédent*** et la sortie ***conclusion*** ou ***conséquent***. Une règle d'inférence est parfois simplement notée $p \rightarrow q$ où p est l'antécédent et q le conséquent.

Pour présenter les langages de règles pour le Web, nous reprenons la classification de Paschke & Boley [2010].

Rule Markup Languages

Un langage de règles est qualifié de *rule markup language* s'il est sérialisé en XML et s'il emploie des URIs/IRIs pour les litéraux et s'il peut s'interfacer avec les langages de représentations des ontologies [Paschke & Boley 2009]. Un *markup language* (***langage de***

18. Il n'y a pas d'objet IFC représentant les portes coupe-feu. Par conséquent, le schéma IFC doit être enrichi pour envisager une utilisation en pratique (voir section 6.1).

balisage) est un type de langage informatique conçu soit pour définir la façon dont les documents doivent être affichés, soit pour montrer explicitement la structure logique et la signification des données d'un document, à l'aide de balises qui annotent ce document [Geroimenko 2004].

— **Rule Markup Language (RuleML)** est un langage de balisage basé sur XML et qui rend possible le stockage, l'échange, la recherche et l'utilisation des règles pour le Web [Geroimenko 2004]. Parmi les applications de RuleML, on trouve, l'écriture de requêtes, les inférences sur des ontologies ou encore l'établissement de correspondances entre ontologies [Geroimenko 2004] [Boley *et al.* 2010]. Par ailleurs, ajoutons que RuleML supporte la ***négation forte*** et la ***négation par l'échec***.

— **Semantic Web Rule Language (SWRL)** est un langage de règles combinant les langages OWL DL et RuleML [Horrocks *et al.* 2004]. Les règles en SWRL sont de la forme $p \rightarrow q$ avec p et q formés d'une conjonction de prédicats unaires ou binaires. De base, SWRL n'est pas un langage décidable, mais peut l'être si on s'assure de manipuler des ***individus nommés***. De plus SWRL n'intègre ni la négation forte, ni la négation par l'échec.

— Le **Rule Interchange Format (RIF)** est une recommandation du *World Wide Web Consortium* (W3C) visant à permettre l'échange des règles entre des systèmes manipulant des règles sur le Web Kifer & Boley [2013]. Dans son dialecte de base, c'est-à-dire le RIF Basic Logic Dialect (RIF-BLD), RIF ne fournit pas la possibilité d'exprimer la négation, qu'elle soit forte ou par l'échec. Par contre, certaines extensions de RIF-BLD permettent d'utiliser la négation forte et la négation par l'échec. C'est le cas du RIF Core Answer Set Programming Dialect (RIF-CASPD) [Heymans & Kifer 2009].

Négation forte et Négation par l'échec

La ***négation forte*** permet d'affirmer qu'un fait est faux si on peut démontrer en un temps fini qu'il est ***explicitement*** faux.

La ***négation par l'échec*** est une forme de négation dans laquelle, un fait est considéré faux si on échoue à montrer qu'il est vrai. Autrement dit, en plus des faits dont on peut prouver explicitement qu'ils sont faux, les faits inconnus sont aussi considérés comme étant faux.

Non-Markup Rule Languages

Contrairement aux langages de règles de types *markup language*, les langages classés comme étant des *Semantic Web Rule Languages* par Paschke & Boley [2010], ont été mis sur pied pour être compatibles avec les langages de représentation de données du Web Sémantique (RDF, RDFS, OWL) et pour avoir une syntaxe facilement lisible par les utilisateurs (en anglais *human-readable syntax*).

— **Notation3** ou **N3** est une notation abrégée, non-XML, de données RDF [Berners-Lee & Connolly 2011]. N3 permet en plus de représenter des règles basées sur des données RDF. Notons que N3 ne permet pas d'exprimer la négation.
— **Prova** est à la fois un langage de règles compatible avec le Web Sémantique et un moteur de règles [Kozlenkov *et al.* 2006]. En tant que langage, il combine les paradigmes des programmations imperative et declarative via une syntaxe semblable à Prolog [19] qui permet de faire appel à des fonctions Java. En tant que moteur d'inférence, il permet d'exécuter des *reaction rules*, de faire de l'*event processing*, d'effectuer des inférences dans des architectures distribuées, etc. Notons que c'est un langage dans lequel il est possible d'exprimer la négation forte et la négation par l'échec.
— **SPARQL Protocol and RDF Query Language (SPARQL)** est la recommandation du W3C pour l'interrogation de données RDF [Prud'hommeaux & Seaborne 2007]. En tant que langage de requête, il est à RDF ce que le langage SQL (*Structured Query Language*) est aux bases de données relationnelles. Pour plus d'information sur le langage SPARQL, se référer à la page 45.

Quel langage de représentation pour nos règles ?

Les récents travaux de Paschke & Boley [2010] permettent d'avoir un aperçu des langages de représentations formelles des règles. À partir de leurs travaux, on a une vue globale de la syntaxe, de l'expressivité de ces langages les uns vis-à-vis des autres, des types de règles (cf. section 3.1.2) que ces langages peuvent modéliser, de leur compatibilité avec les langages du Web Sémantique, de leurs applications, etc. Pour rappel, nous nous intéressons uniquement aux ***derivation rules*** dans le cadre de notre travail de thèse (cf. section 3.1.5). À la figure 3.6, on peut voir une même exigence représentée en SPARQL et SWRL. Si les différences d'ordre syntaxique entre ces représentations sont flagrantes, on peut toutefois noter un point commun : ***la présence des mêmes entités - issues de l'ontologie, et des mêmes fonctions de comparaisons***. Pour mettre en lumière ces similitudes, nous avons représenté les assertions similaires entre les expressions de la règle en SPARQL et SWRL, à l'aide d'une même couleur.
On peut ainsi conclure que le principal défi à relever par notre méthode de formalisation automatique est de pouvoir :

1. Identifier les entités formelles nécessaires à la formalisation de l'exigence. Dans cet exemple, ce sont `:PlateformeElevatrice`, `:vitesseNominale`, `:NonConforme` et l'opérateur de comparaison
2. Agencer correctement les variables et les littéraux pour chaque entité
3. Identifier les entités portant les marques de négations

19. https ://fr.wikipedia.org/wiki/Prolog

```
CONSTRUCT {
  ?plf rdf:type :NonConforme }
WHERE {
   ?plf rdf:type :PlateformeElevatrice.
    OPTIONAL { ?plf :vitesseNominale ?vit
               FILTER (?vit <= 0.15) }
    FILTER (!bound(?vit)) }
```

(a) Exemple de règle en SPARQL. La partie `WHERE` correspond à l'antécédent et la partie `CONSTRUCT` au conséquent de la règle. Pour plus de détails sur SPARQL voir page 45.

```
:PlateformeElevatrice(?plf), :vitesseNominale(?plf, ?vit),
    swrlb:greaterThan(?vit, 0.15) -> :NonConforme(?plf)

:PlateformeElevatrice(?plf), (:vitesseNominale max 0)(?plf)
   -> :NonConforme(?plf)
```

(b) Exemple de règle en SWRL (*human-readable syntax*). Le préfixe `swrlb` renvoie au *namespace* `http://www.w3.org/2003/11/swrlb#`. L'antécédent et le conséquent sont séparés par le symbole `->`.

FIGURE 3.6 – Expressions formelles de la prescription ***"La vitesse de la plate-forme ne doit pas dépasser 0,15 m/s"*** en SPARQL et SWRL.

4. Identifier les entités à mettre respectivement dans l'antécédent et le conséquent de la règle

En suivant ces étapes, on aboutit à une "***règle temporaire***", ***indépendante du langage formel*** et qu'il faut exprimer dans le langage de son choix. Pour notre exemple, cette règle temporaire peut être représentée de la façon suivante [20] :
`:PlateformeElevatrice(plf)` $\wedge$ `:vitesseNominale(vit)` $\wedge$ `!lessThanOrEqual(vit,0.15)` $\rightarrow$ `NonConforme(plf)`.
Par conséquent, on peut proposer une méthode de formalisation indépendante du langage formel. Toutefois, pour aller jusqu'au bout du processus de formalisation, nos règles formelles seront représentées en SPARQL. Aussi, en plus de la brève description de SPARQL faite ci-dessus, nous fournissons un supplément de détails sur ce langage dans les paragraphes ci-dessous (voir page 45). Pour représenter nos règles formelles, il suffit de disposer d'un langage de représentation de règles permettant d'exprimer la négation par l'échec. En effet, nous travaillons avec des règles métiers ; ces dernières devant nous permettre de détecter des éléments non-conformes, c'est-à-dire qui enfreignent la norme. Or, dans bien des cas, un élément sera jugé non-conforme parce qu'il ***ne respectera pas*** certaines conditions. Cette notion de non-respect se traduit de manière formelle pas une négation et précisément par une négation par l'échec, dans le cas des règles métiers. Dire d'un élément qu'il ne présente pas une caractéristique requise, c'est

20. Cette syntaxe est généralement utiliser dans la littérature pour représenter les règles indépendamment de tout langage formel [Horrocks & Patel-Schneider 2004, section 3.2].

(1) soit prouver explicitement qu'il ne présente pas cette caractéristique, (2) soit ne pas réussir à obtenir (la valeur de) cette caractéristique pour cet élément. Conclure à une non-conformité dans le cas (1) est immédiat. Dans le cas (2) nous concluons à une non-conformité parce que l'information est manquante. Ne pas déduire une non-conformité dans ce second cas serait équivalent à conclure que l'élément contrôlé est conforme. Or, étant donné que les règles métiers nous permettent d'attester du respect des conditions de sécurité des personnes et des biens, ne nous pouvons pas conclure à une conformité en cas d'information(s) manquante(s). Nous pourrions donc exprimer nos règles formelles en utilisant RuleML, RIF-CASPD, Prova ou SPARQL.
Dans le cadre de notre thèse, nous avons choisi SPARQL. Bien que n'étant pas à proprement parler un langage d'écriture de règles, des chercheurs ont pu développer des moteurs de règles, supportés par des requêtes SPARQL (par exemple CORESE [Corby *et al.* 2006]). Par ailleurs de tels moteurs de règles ont pu être éprouvés dans de nombreux travaux de recherches [Corby *et al.* 2006], [Yurchyshyna 2009], [Bouzidi 2013]. De plus, c'est un langage populaire au sein de la communauté du Web Sémantique et il dispose d'une syntaxe *human-readable*. En outre, SPARQL présente l'avantage pour nous, d'assurer la continuité avec les travaux de thèses précédents ([Yurchyshyna 2009] et [Bouzidi 2013]).

SPARQL Protocol and RDF Query Language

SPARQL est la recommandation du W3C pour l'interrogation de données RDF [Prud'hommeaux & Seaborne 2007]. En tant que langage de requête, il est à RDF ce que le langage SQL (*Structured Query Language*) est aux bases de données relationnelles.
RDF permettant de représenter des données sous la forme de triplets (⟨ sujet, predicat, objet ⟩), l'énoncé de base du langage SPARQL est lui aussi un triplet. Effectuer une interrogation en SPARQL consiste à écrire un ***graphe requête*** pour lequel on recherche des occurrences dans un graphe cible, autrement dit c'est un appariement de graphe (*graph matching*). Le calcul d'une requête revient ainsi, à rechercher tous les sous-graphes, correspondant au patron dénoté par le graphe fourni dans la requête. Cette recherche de correspondance peut déboucher sur zéro, un ou plusieurs résultats.
Écrite dans une syntaxe proche de Turtle, une requête SPARQL peut être de quatre types : `SELECT` (pour sélectionner des variables et leurs valeurs), `ASK` (requête booléenne dont la valeur est ***true*** s'il y a au moins une réponse et ***false*** sinon), `DESCRIBE` (pour la description de ressources), `CONSTRUCT` (retourne un résultat sous forme de graphe). C'est ce dernier type de requête qui se prête à l'écriture de règles sous forme de requêtes.
Une requête SPARQL `CONSTRUCT` peut être illustrée par le schéma suivant :
Sur ce schéma, on retrouve :

- Une partie déclaration qui sert à lister les espaces de nommage. Chaque espace est introduit à l'aide du mot clé `PREFIX`
- Une partie, introduite par le mot clé `CONSTRUCT`, qui fourni le ***patron*** (*template*)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://example.com/ifc#>

CONSTRUCT {
  ?plf rdf:type :NonConforme }

WHERE {
   ?plf rdf:type :PlateformeElevatrice.
    OPTIONAL { ?plf :vitesseNominale ?vit.
               FILTER (?vit <= 0.15) }
    FILTER (!bound(?vit)) }
```

Déclaration des schémas

Graphe résultat

Contraintes sur les variables

Code de couleurs: MOTS CLÉS et fonctions prédéfinies - variables - Entités et litéraux

du graphe résultat attendu, sous la forme d'une conjonction de triplets. Dans ces derniers on trouve des ***variables*** qui seront remplacées par leur valeur pour constituer le graphe résultat de la requête. Les variables se reconnaissent au point d'interrogation (?) qui les précède.

— Les sous graphes, sous la forme d'une conjonction de triplets, à respecter par les variables du graphe résultat. Ces contraintes sont contenues dans un bloc `WHERE`. Ce bloc peut contenir :
  — des sous-blocs `FILTER` pour ajouter des contraintes. C'est le cas des comparaisons ou des fonctions prédéfinies telles que `bound` - qui permet de savoir si oui ou non une variable est instanciée.
  — des sous-blocs `OPTIONAL` pour rendre optionnel le ***matching*** d'un certain nombre de triplets.

L'exemple ci-dessus peut se traduire par : "Construisez le graphe des éléments `?plf` qui sont non-conformes. Ces éléments `?plf` doivent êtres des plateformes élévatrices dont la vitesse n'est pas inférieure ou égale à 0.15 ou dont la vitesse n'est pas fournie". Depuis la version 1.1 de SPARQL de nouveaux opérateurs plus simples sont disponibles pour exprimer la négation ; ce sont `MINUS` et `NOT EXISTS`[21]. L'exemple ci-dessus est équivalent à :

```
CONSTRUCT{?plf rdf:type :NonConforme}
WHERE{?plf rdf:type :PlateformeElevatrice.
       MINUS{ ?plf :vitesseNominale ?vit. FILTER(?vit <= 0.15)} }
```

Nous venons d'achever la présentation des langages de représentation formelle des règles. À présent, nous nous penchons sur les travaux de la littérature qui sont relatifs à la formalisation automatique des règles.

21. https ://www.w3.org/TR/sparql11-query/#neg-notexists-minus

3.3 Formalisation automatique de règles métiers

Dans le chapitre 2, sur le contexte de notre travail de thèse, nous avons souligné l'importance que revêt la mise sur pied d'une méthode automatique de formalisation des règles métiers. L'enjeu principal est de pouvoir disposer d'un dépôt de règles formelles, compatibles avec les maquettes numériques des bâtiments, et pouvant permettre de révéler les défauts de ces bâtiments, dès la phase de conception. Disposer d'une version formelle d'exigences réglementaires, nécessite de pouvoir surmonter certains écueils que nous avons présentés à la section 3.1. En s'appuyant sur des résultats de notre travail, exposés dans [Kacfah Emani 2014], nous reprenons brièvement ces défis à résoudre sous forme de fonctionnalités à implémenter par les approches de formalisation de règles :

($\mathcal{F}_1$) L'isolation des exigences réglementaires atomiques dans une prescription réglementaire complexe, c'est-à-dire contenant plusieurs exigences

($\mathcal{F}_2$) La détermination du contexte d'application des règles métiers

($\mathcal{F}_3$) La gestion des inflexions (grammaticales et autres[22]) des termes dans les exigences

($\mathcal{F}_4$) La résolution des paraphrases

($\mathcal{F}_5$) La résolution des co-références et des anaphores

($\mathcal{F}_6$) La détermination de l'antécédent et du conséquent de la règle

($\mathcal{F}_7$) La mise en évidence des paramètres implicites (par exemple dans une exigence comme "l'entrée doit être suffisamment large" il faut faire ressortir la caractéristique ***largeur minimale*** de l'entrée.)

À ces fonctionnalités, nous ajoutons ($\mathcal{F}_8$) relative à l'élagage des expressions non pertinentes (voir section 3.1.3 page 27).

Dans cette section, nous allons explorer la littérature sur la formalisation des règles en présentant succinctement les travaux sur ***la détection et la réécriture des règles métiers***, suivi d'approches de ***formalisations de règles***. Par la suite nous regarderons de manière détaillée deux approches de formalisation automatique de règles métiers.

3.3.1 Détection et reformulation de règles métiers

Dans leurs travaux sur les politiques de sécurité, Brodie *et al.* [2006], Reeder *et al.* [2007] nous proposent des recommandations pour faciliter l'écriture et la compréhension des règles métiers. Comme nous l'avons mentionné depuis les premiers chapitres, nous travaillons sur des corpus déjà existants. Ainsi il revient dans cette tâche de cerner les règles à l'intérieur des textes, de les débarrasser de termes non-pertinents et de les mettre sous une forme interprétables par les machines. En effet, avec Graham [2006], nous rappelons qu'une règle métier se doit d'être ***atomique***, ***compacte***, ***bien formée***, ***non***

22. Un terme peut, par exemple, apparaître de manière partielle ou légèrement différentes dans les textes réglementaires ("plate-forme" pour désigner "plate-forme élévatrice", "handicapé" pour faire référence à "personne handicapé", "personne à faible mobilité" pour mentionner "personne à mobilité réduite", etc.)

ambiguë et ***construite sur un vocabulaire métier***. Cette tâche de compréhension automatique des règles est parfois considérée dans la littérature comme étant impossible, ou tout au moins très complexe [Lévy & Nazarenko 2013].

Proposer l'ensemble des fonctionnalités ($\mathcal{F}_1$)-($\mathcal{F}_8$) listées ci-dessus n'est pas une tâche simple, mais il existe des travaux ayant abordés certains défis sous-jacents à celles-ci, permettant ainsi d'offrir une aide précieuse aux experts métiers. Ainsi, dans [Maxwell & Anton 2009], une approche détaillée, mais manuelle, est proposée pour identifier et extraire et formaliser des règles. Une application à la main, de cette méthodologie ne permet pas d'être en mesure d'apprécier la difficulté de réalisation des fonctionnalités ($\mathcal{F}_3$)-($\mathcal{F}_8$). Se rapprochant d'une méthode automatisable, Breaux & Anton [2005] tirent profit du *goal mining* ([Anton 1997, chap. 4]), pour la gestion des politiques de sécurité. Ils reformulent les déclarations d'objectifs sous forme de ***déclarations atomiques*** (*Restricted Natural Language Statements*). Ces dernières sont des phrases contenant un ***agent***, l'***action*** qu'il doit réaliser et le ***but*** visé par cette action. Bien que débouchant sur des expressions formelles, cette approche est appliquée manuellement. Par conséquent, très peu de fonctionnalités, parmi celles listées ci-dessus, sont prises en compte. Lévy & Nazarenko [2013] nous proposent une architecture détaillée et des procédures à mettre en oeuvre pour la mise sur pied d'un outil complet (pas de fonctionnalités ($\mathcal{F}_2$) et ($\mathcal{F}_8$)) de formalisation de règles. Ils proposent dans un premier temps de construire une ontologie de domaine, en utilisant l'outil d'extraction de termes TERMINAE [Aussenac-Gilles *et al.* 2008], à partir des textes réglementaires. Puis ils bâtissent un index ([Guissé *et al.* 2011]) permettant de relier chaque paragraphe issu des textes réglementaires, avec les entités issues de l'ontologie de domaine et avec les règles extraites manuellement de ce paragraphe. Ces règles sont encore sous une forme semi-formelle, à savoir ***Semantics of Business Vocabulary and Business Rules - Structured English*** (SBVR-SE -voir page 65). L'édition de ces règles est faite manuellement, mais l'utilisateur est aidé par un outil approprié [Lévy *et al.* 2010].

Dans le registre des approches automatiques permettant la reformulation de règles, nous avons les travaux de Bajwa *et al.* [2011]. Dans cet article, Bajwa, Lee, & Bordbar proposent une méthode de transformation d'exigences du langage naturel vers des déclarations conformes au langage naturel contrôlé SBVR-SE. Dans leur application, la terminologie de domaine est capturée par un diagramme de classes UML [23]. Sur leur corpus, leur approche obtient une précision de 87.33%. Toutefois les difficultés inhérentes à ($\mathcal{F}_4$), ($\mathcal{F}_5$), ($\mathcal{F}_7$) et ($\mathcal{F}_8$) ne sont pas prises en compte dans leur approche. De plus, le langage SBVR-SE n'exigeant pas que l'écriture d'une règle mette en évidence des prémisses et une conclusion, la fonctionnalité ($\mathcal{F}_6$) n'est pas native dans leur approche.

23. http ://www.uml.org/

3.3.2 Approches de formalisation de règles métiers

Les approches présentées à la section précédente, exception faite de l'approche de Breaux & Anton [2005], permettent d'obtenir des expressions semi-formelles. Dans cette section, nous allons aborder des approches dont le but est d'obtenir une version formelle et exécutable des règles métiers.

Nous avions mentionné que dans leurs travaux sur la formalisation des politiques de sécurité, Breaux & Anton [2005] reformulent les exigences en un ensemble de phrases courtes et bâties autour d'un ***agent***, d'une ***action*** et d'un ***objet***. Cela leur permet de stocker les exigences sous une forme de table à trois colonnes. Une telle représentation ne peut pas convenir à tous les types d'exigences. D'autres approches ont fait le choix des langages du Web Sémantique pour l'expression des règles. Comme nous l'avons énoncé à la section 3.2, ces langages ont en leur faveur les avantages de l'expressivité, de l'interopérabilité, de la standardisation, etc. Parmi ces approches, [Yurchyshyna *et al.* 2008a;b] présentent les premières démarches de formalisation des codes de la Construction sous formes de requêtes SPARQL. Cette démarche sera par la suite reprise par Bouzidi *et al.* [2012]. Toujours dans le domaine de la Construction, on retrouve les travaux de Pauwels *et al.* [2011b;a] où les exigences sont modélisées sous forme de règles respectant le formalisme N3Logic. Toutefois, ces méthodes de formalisation de normes demeurent manuelles.

Cependant, il existe dans la littérature des travaux qui de bout en bout, permettent de transformer automatiquement des exigences, initialement en langage naturel, en des expressions formelles. Nous les présentons en détails dans les sections suivantes.

Méthodes automatiques de formalisation de règles métiers

Les travaux de Hassanpour, O'Connor, & Das [2011]. Dans leur travaux sur les règles, Hassanpour, O'Connor, & Das proposent une méthode automatique de formalisation d'exigences. Ici, le langage utilisé pour écrire les règles formelles est SWRL. Pour disposer de l'expression formelle d'une exigence donnée, leur méthode prend en entrée :

- Ladite ***exigence réglementaire en langage naturel***. Cette dernière est fournie par l'utilisateur qui a pris soin de la sélectionner dans un texte normatif.
- Une ***ontologie décrivant le domaine concerné***. Notons que pour les cas d'application de leur méthode, à savoir les conditions de locations de voiture dans des agences, l'ontologie de domaine est construite en ***reprenant directement les termes apparaissant dans la politique de location des agences***.
- Un ***ensemble d'exigences déjà disponibles sous forme de règles SWRL***. Ce dépôt pré-existant, contenant des règles SWRL, leur permet de connaître la forme que peut revêtir le conséquent de l'exigence réglementaire en cours de formalisation. Dans leur auto-critique, les auteurs notent ce point nécessite d'être résolu pour améliorer leur méthode.

L'algorithme de formalisation proposé par Hassanpour *et al.* [2011] se compose de sept (07) étapes que nous illustrons de manière détaillée ci-dessous.
Pour cette illustration, nous allons prendre l'exigence "*A lifting platform must be composed of a safety control device*". Comme ontologie de domaine, nous allons nous référer à l'ontologie du *listing* 3.3 ci-dessous.

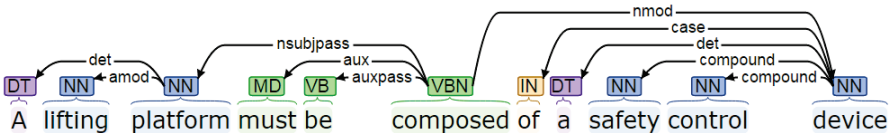
Listing 3.3 – Ontologie pour l'illustration des approches de Hassanpour *et al.* [2011] et Kang *et al.* [2015]

```
@prefix : <http://example.com/LiftingPlatformOntology#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:LiftingPlatform rdf:type owl:Class; rdfs:label "lifting platform".
:Device rdf:type owl:Class; rdfs:label "device".
:EmergencyControlDevice rdf:type owl:Class;
                        rdfs:label: "emergency control device";
                        rdfs:subClassOf :Device.
:isConform rdf:type owl:DatatypeProperty; rdfs:label "is conform";
           rdfs:domain owl:Thing;
           rdfs:range xsd:boolean.

:isComposedOf rdf:type owl:ObjectProperty; rdfs:label "is composed of";
              rdfs:domain :LiftingPlatform;
              rdfs:range :Device.

:nominalSpeed rdf:type owl:DatatypeProperty; rdfs:label "nominal speed";
              rdfs:domain :LiftingPlatform;
              rdfs:range xsd:double.
# La règle SWRL ci-après est déjà présente dans le dépôt de règles
# :LiftingPlatform(?a) ∧ :nomimalSpeed(?a,?b) ∧ swrlb:lessThan(?b,0.15)
#                                 -> :isConform(?a, "true").
```

Les étapes de cette approche sont les suivantes (rappelons qu'elles sont automatiques) :

1. Les libellées des termes de l'ontologie sont étendus à l'aide de leur synonymes présents dans Wordnet [24] [Miller 1995]. On obtiendra par exemple pour :
   — *device*, les synonymes *gimmick* et *twist* ;
   — *compose* : *write*, *pen*, *indite*, *frame*, *draw up*, *compile* ;
   — *lift* : *elevate*, *raise*, *move up*, *rustle*, *revoke*, etc.
   — *platform* : *chopine*, *program*, etc.
2. Le calcul des dépendances à l'aide du *Stanford parser* [25] permet d'obtenir le graphe ci-dessous :

24. http ://wordnetweb.princeton.edu/perl/webwn
25. http ://corenlp.run/



La liste exhaustive des dépendances fournies par le *Stanford parser* et leur signification est disponible dans le manuel de référence [De Marneffe & Manning 2008].

3. L'identification des propriétés : "*be composed of*" est reconnu comme étant une mention de l'*object property* `:isComposedOf`, présente dans l'ontologie. Notons que Hassanpour *et al.* identifient uniquement des *object properties*. ***Les propiétés de type datatype ne sont pas résolues***.

4. La détermination des classes (concepts) : l'expression "*a safety control device*" est identifiée comme étant une mention de la *class* `:SafetyControlDevice`, présente dans l'ontologie. Il en va de même pour "*a platform*" reconnue comme étant une mention de la classe `:LiftingPlatform`.
   L'identification des propriétés et des classes, s'appuie sur un mesure de similarité entre chaînes de caractères. La mesure utilisée par Hassanpour *et al.* [2011] est la distance de Needleman & Wunsch [1970].

5. L'antécédent de la règle est constitué comme une conjonction ("et" logique) des atomes formellement identifiés. Si la règle contient un "ou", alors la règle originale donnera lieu à autant d'expressions formelles que de disjonctions. Ainsi l'antécédent ***provisoire*** de la règle est :
   `:LiftingPlatform(?a)` ∧ `:SafetyControlDevice(?b)` ∧ `isComposedOf(?a,?b)`.

6. Si les variables `?a` et `?b` de la propriété `:isComposedOf` n'étaient pas typées par des classes, ces variables auraient été typées par les *domain* (`rdfs:domain`) et *range* (`rdfs:range`) de `:isComposedOf` fournis dans l'ontologie. L'antécédent de la règle, obtenu à l'étape précédente, reste inchangé.

7. Pour déterminer le constituant de la règle, Hassanpour *et al.* regardent les conséquents des règles déjà présentes dans le dépôt de règles et s'inspirent de celui dont la formulation se rapproche le plus des termes de l'exigence en cours. L'expression SWRL obtenue au terme de ces étapes est :
   `:LiftingPlatform(?a)` ∧ `:SafetyControlDevice(?b)` ∧ `:isComposedOf(?a,?b)` **`-> :Conform(?a, "true")`**.

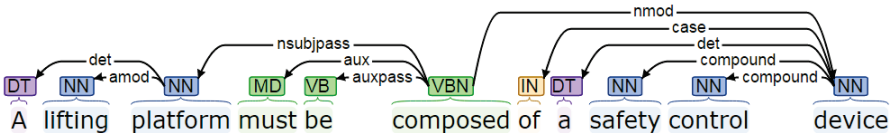
Les travaux de Kang, Patil, Rangarajan, Moitra, Jia, Robinson, & Dutta [2015]. Kang *et al.* [2015] ont travaillé avec des règles pour la fabrication de pièces mécaniques. À la différence de [Hassanpour *et al.* 2011], ils s'appuient sur le langage SADL [26] (*Semantic Application Design Language*) pour la représentation des données. Par ailleurs, leur approche prend en entrée :

26. http ://sadl.sourceforge.net/

— Un ***texte*** contenant des exigences réglementaires,
— Une ***ontologie de domaine***.

Nous constatons que par rapport à l'approche précédente, il n'est pas nécessaire de disposer d'un échantillon de règles précédemment formalisées, pour pouvoir formaliser automatiquement de nouvelles exigences. De plus, ici nous avons un texte et pas une phrase en entrée. Les étapes de leur approche sont les suivantes (pour l'illustration, nous reprenons l'exigence "*A platform must be composed of a safety control device*" et l'ontologie du *listing* 3.3) :

1. Le ***pré-traitement du texte***. Cette étape permet de garder dans le texte les phrases susceptibles de contenir des exigences. Cette étape conservera les énoncés contenant des mots-clés tels que *must*, *shall/should*, *when*, *if . . . then*, etc. La phrase utilisée pour illustration passera donc ce filtre avec succès.
2. L'***analyse syntaxique et grammaticale*** de l'exigence. Cette étape comprend, dans l'ordre :
   (a) La ***tokenization*** qui permet d'isoler chaque mot et chaque signe de ponctuation présent dans l'exigence.
   (b) L'***étiquettage morpho-syntaxique*** qui permet d'obtenir la catégorie grammaticale d'un mot. Par exemple : ***déterminant*** (DT), ***nom singulier*** (NN), ***verbe à l'infinitif*** (VB), ***verbe au participe passé*** (VBN), etc. Les labels de ces catégories dépendent des analyseurs [27].
   (c) La ***détermination des dépendances syntaxiques***. Cette étape est équivalente à l'étape 2 de l'approche proposée par Hassanpour *et al.* [2011], à l'exception près de la résolution de co-référence -sans résultat pour cet exemple, et permet donc d'obtenir le même résultat, que nous reprenons ci-dessous.

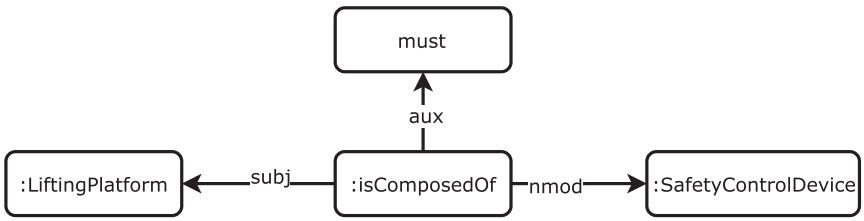


   (d) La ***résolution de co-références***. Cette étape permet de révéler les mots auxquels font références les pronoms éventuels, présents dans les exigences.
3. La ***détermination du graphe des termes***. Nous pouvons décomposer cette étape de la manière suivante :
   (a) L'***identification des termes de l'ontologie*** présents dans la phrase. Ce processus permet de résoudre les entités `:SafetyControlDevice`, `:isComposedOf`. Notons que dans cette approche, l'appariement entre les termes de l'exigence et les libellés des entités de l'ontologie se fait que si les chaînes de caractères sont les mêmes. Ainsi l'expression "*platform*" n'est pas identifiée comme étant

27. Ceux employés ici sont ceux du Stanford parser qui les reprend du *treebank* - corpus annoté - de l'université de Pennsylvanie et plus connu sous le nom de *Penn Treebank* `http://www.cis.upenn.edu/~treebank/`.

une mention de l'entité `:LiftingPlatform` et l'approche est interrompue. Pour pouvoir continuer à dérouler et illustrer cette approche, nous que l'exigence eut été "*A **lifting platform** must be composed of a safety control device*" et donc que l'entité `:LiftingPlatform` soit identifiée.

(b) La ***détermination des relations entre les termes***. En s'appuyant sur les termes obtenus précédemment et sur certains "mots clés" (voir étape suivante), les marqueurs de négation éventuels, et sur les dépendances syntaxiques, Kang *et al.* [2015] obtiennent un graphe dit de ***dépendance des termes***. Pour notre exemple, ce graphe est le suivant :



4. L'***obtention de la règle proprement dite***. Pour cette dernière opération, plusieurs cas existent et dépendent de certaines "expressions clés" contenues dans la phrase :
   — Soit la règle est de la forme ***si ... alors***, auquel cas, l'antécédent est constitué des termes contenus dans la proposition introduite par ***si*** et le conséquent dans celle ouverte par ***alors***.
   — Soit la règle n'est pas de la forme précédente mais contient un ***doit*** (en anglais *must/shall/should*) alors la proposition supportée par ce mot induit le conséquent de la règle et tout le reste de l'exigence permet d'obtenir l'antécédent.
   — Soit la règle est introduite par un ***lorsque*** (en anglais *when/given*) alors la proposition qui suit ces mots sera utilisée pour déterminer les prémisses de la règle et le reste de l'exigence sera exploitée pour la conclusion, etc.

   Au terme de la détermination de l'antécédent et du conséquent, il peut être nécessaire de restreindre le *domain* et le *range* des propriétés. Pour notre exemple, la règle obtenue est :

   `:LiftingPlatform(?a)` $\wedge$ `:SafetyControlDevice(?b) -> isComposedOf(?a, ?b)`

   L'expression ainsi obtenue peut se traduire par "*if ?a is a lifting platform and ?b is a safety control device then ?a is composed of ?b*". On remarque qu'elle n'est pas fidèle à l'exigence de départ. Par ailleurs, formuler de cette manière, on ne peut pas attester de la conformité ou de la présence d'un défaut sur une plateforme élévatrice.

Bilan de la comparaison des deux approches ([Hassanpour *et al.* 2011] et [Kang *et al.* 2015]).

1. **Données d'entrées.** Toutes les deux approches s'appuient sur une ***ontologie de domaine existante***. Hassanpour *et al.* [2011] prennent directement en entrée la règle (une phrase) à formaliser alors que Kang *et al.* [2015] prennent un texte (ensemble

de phrases) à l'intérieur duquel ils cibleront automatiquement les phrases pouvant contenir des règles. Notons qu'aucune information statistique n'est fournie sur la qualité de ce processus de détection automatique de prescriptions dans un texte. Contrairement à l'approche de [Kang *et al.* 2015], un ***dépôt initial de règles formelles*** est nécessaire pour mettre en oeuvre la méthode proposée par Hassanpour *et al.* [2011].

2. **Analyse des exigences.** Les deux approches reposent sur les tâches de base du traitement automatique du langage naturel : *tokenization*, étiquetage morpho-syntaxique, détermination des dépendances grammaticales. Toutefois, l'approche de Kang *et al.* [2015] présente l'avantage de résoudre les co-références des éventuels pronoms (dans les limites de l'outil utilisé - le Stanford parser).

3. **Alignement sur l'ontologie de domaine.** Les deux approches utilisent les entités de l'ontologie dont les libellés se rapprochent le plus des mots utilisés dans l'exigence. Aucune utilisation n'est faite des mots de l'exigence qui ne se retrouve pas dans l'ensemble des libellés des entités de l'ontologie de domaine. Néanmoins, dans [Hassanpour *et al.* 2011], un recours est fait à une ressource lexicale externe, Wordnet, pour gérer les possibles cas de synonymie. Cette utilisation de Wordnet présente toutefois l'inconvénient d'être ***non contextualisée***. Pour illustration, Hassanpour *et al.* [2011] utilise le mot *write* comme synonyme de *compose*, alors que dans le domaine conceptualisé par l'ontologie proposée en exemple, *write* et *compose* ne peuvent pas jouer des rôles similaires. En outre, Hassanpour *et al.* sont capables d'aligner les termes de l'exigence sur des termes jugés proches, de l'ontologie de domaine. Or Kang *et al.* supposent que les termes de l'exigence apparaissent exactement tels quels dans la liste des termes de l'ontologie. Autrement dit, ils n'implémentent pas la normalisation syntaxique (cf. section 3.1.3 page 26). Pour finir, notons que l'approche de Hassanpour *et al.* [2011] ne résout que les *object properties* contrairement à celle de Kang *et al.* [2015] qui peut identifier les propriétés dont les valeurs sont des littéraux (entiers, flottants, booléens, etc.).

4. **Assemblage des composants de la règle.** Chacune des deux approches exploitent les dépendances grammaticales et les termes issues de l'alignement avec l'ontologie pour déterminer les relations entre les variables associées à chaque terme. De plus, dans les deux approches on observe un recours au *domain* et au *range* des propriétés pour restreindre le champ des variables. Des différences existent cependant entre les deux approches pour la stratégie de détermination du conséquent de la règle. Hassanpour *et al.* [2011] calquent le conséquent de l'exigence courante sur les conséquents des règles déjà présentes dans le dépôt. De plus ils s'assurent que ce conséquent n'introduit pas de variables non présentes dans l'antécédent de la règle. Quant à l'approche de Kang *et al.* [2015], elle obtient le conséquent par une analyse directe de l'exigence proprement dite. Par ailleurs, il est important de noter que dans l'une et l'autre approche, le ***but*** de la règle n'influe pas le conséquent.

En effet, comme nous l'avons souligné à la section 3.1.5, les règles fournies dans les textes normatifs permettent de certifier qu'un produit ne présente pas de défauts. Ainsi lorsqu'on s'attarde, par exemple, sur la règle formelle obtenue en déroulant l'algorithme proposé dans [Kang *et al.* 2015], on se rend compte que la conclusion, "*a lifting platform is composed of a safety control device*", permet de déduire qu'une plateforme possède un équipement de sécurité. Or, au regard de l'exigence, du point de vue de l'expert métier, on s'attend à la conclusion "*A lifting platform is not conform*" - if "*this platform does not have a safety control device*". Un regard similaire porté sur l'approche de Hassanpour *et al.* [2011] montre que le conséquent de la règle dépend des conséquents des règles présentes dans le dépôt. Ainsi, le seul moyen de garantir l'obtention d'une règle de non-conformité est de disposer dans le dépôt d'exemples de règles de non-conformité.

Vis-à-vis des fonctionnalités ($\mathcal{F}_1$)-($\mathcal{F}_8$) listées en introduction de cette section (voir page 47), on peut relever que :

— L'approche de Hassanpour *et al.* [2011] implémente les fonctionnalités ($\mathcal{F}_1$), ($\mathcal{F}_3$), ($\mathcal{F}_4$) et ($\mathcal{F}_6$).

— Dans les travaux de Kang *et al.* [2015], les fonctionnalités ($\mathcal{F}_5$) et ($\mathcal{F}_6$) sont implémentées.

On constate ainsi que les fonctionnalités ($\mathcal{F}_2$) de détermination du contexte d'une règle et ($\mathcal{F}_7$) de mise en évidence des attributs implicites ne sont pas supportées par les méthodes existantes. Il en va de même de l'élagage - ($\mathcal{F}_8$).

Pour clore ce bilan, notons que le matériel de test de ces approches est de taille réduite et indisponible. Par exemple, les ontologies de domaine utilisées ou les implémentations des approches ne sont pas exploitables. Il en est de même des exigences réglementaires utilisées. L'évaluation du travail de Hassanpour *et al.* [2011] est effectuée sur onze (11) exigences. Toutefois, la ***précision*** de leur méthode d'alignement, entre les exigences et leur ontologie, est de 100% et le ***rappel*** de 94%. Ces chiffres concernent uniquement les neuf (09) exigences dont l'expression formelle est conforme à l'esprit de la prescription originale, d'après les auteurs. Le corpus utilisé pour l'appréciation du travail de Kang *et al.* [2015] n'est pas fourni et ses caractéristiques ne sont pas renseignées par les auteurs.

Nous avons ouvert ce chapitre par une présentation des différentes classifications des règles. Ensuite nous avons fait un exposé des langages de représentation formels de règles. Enfin, nous avons présenté les méthodes permettant de transformer des règles, écrites en langage naturel, en des expressions formelles. En introduisant ce travail de thèse, nous avons insisté sur l'importance de la transformation automatique des règles, de leur forme originale, en des expressions exécutables. De même, nous avons mentionné qu'il est vital, que l'expert métier, en tant que garant de la qualité et du bon déroulement du contrôle de conformité, devait avoir une vue sur les règles formelles. Or, ce dernier n'est pas, a priori, expert en représentation des connaissances. Dans cette

dernière partie de l'état de l'art, nous discutons de la pertinence des langages naturels contrôlés (LNC) pour permettre à l'expert métier d'avoir le contrôle sur les règles formelles. L'importance des LNC étant acquise, nous présentons un état de l'art sur les LNC proprement dits.

3.4 Langages Naturels Contrôlés

3.4.1 De l'importance des langages naturels contrôlés

Notre travail de thèse prend racine dans le contrôle automatique de la conformité réglementaire dans le domaine de la Construction. Nous avons mentionné le fait que rendre automatique le contrôle de la conformité réglementaire nécessite que des ***personnes*** fournissent les normes, aujourd'hui en langage naturel, sous une forme processable. Les personnes les plus compétentes pour réaliser ce travail sont celles qui possèdent une très bonne connaissance du domaine de la Construction et une excellente compréhension des textes de loi qui régissent ce domaine. Nous appelons ces personnes les ***experts métiers*** [28]. Cependant les experts métiers ne sont pas nécessairement des informaticiens et/ou des logiciens. Pourtant ces derniers doivent approuver la justesse des prescriptions réglementaires écrite dans un langage formel. Et s'il s'avérait qu'une prescription formelle soit partiellement ou complètement erronée, les experts doivent pouvoir la modifier voire la réécrire de bout en bout.

Nous voyons ainsi que les experts métiers, qui ne parlent qu'en langue naturelle enrichie du jargon de leur domaine d'expertise, doivent pouvoir ***comprendre*** et ***écrire*** des expressions formelles ; ces dernières devant nécessairement être en parfait accord avec les textes réglementaires originaux. Toutefois, les experts métiers ne connaissent pas, a priori, les langages formels et n'ont pas vocation à acquérir cette compétence.

Pour résoudre ce problème, trois (03) solutions principales sont envisageables par les experts métiers :

1. Travailler avec des logiciens à qui les experts expliqueront chaque exigence réglementaire. Les logiciens fourniront ainsi une version formelle des exigences selon ce qu'eux, et non les experts métiers, ont compris. Les experts métiers sont obligés dans ce cas de faire entièrement confiance à la compréhension de leurs partenaires experts en logique. C'est la solution adoptée dans [Yurchyshyna 2009].
2. Réécrire les exigences réglementaires sous une forme "plus simple" et sans éléments implicites. Si les exigences réglementaires simplifiées restent en langage naturel, elles sont (a priori) compréhensibles par toute personne et donc par des

28. Dans notre contexte d'application, experts métiers sous-entend "du domaine de la Construction". Toutefois, Quel que soit le domaine d'application, on retrouve des personnes expertes et fiables en termes de connaissances métiers.

logiciens qui peuvent les mettre sous une forme formelle. Toutefois, si l'expert métier veut mettre à jour une exigence, il doit de nouveau la reformuler et refaire appel à un logicien pour la formalisation. Cette solution a, par exemple, été adoptée dans les travaux de Bouzidi [2013].

3. Rédiger les exigences en ***totale autonomie***, mais en troquant le langage naturel pour un autre langage. Cet autre langage peut être un ***langage naturel contrôlé*** (voir définition et doit avoir la caractéristique essentielle d'être ***automatiquement transformable dans un*** **langage formel** .

Parmi les trois possibilités ci-dessus, nous affirmons que la troisième est la plus indiquée. En effet, elle garantit que la justesse des expressions formelles obtenues à partir des exigences réglementaires soit toujours entre des mains expertes. Par ailleurs, l'expert métier peut à tout moment et en toute indépendance modifier ou supprimer toute ou partie de l'expression formelle (via le langage contrôlé), avec l'assurance que celle-ci est toujours correcte. Ce rôle axial, joué par les langages contrôlés, est représenté dans la figure 3.7. Dans les sections qui suivent, en nous appuyant sur l'état de l'art de Kuhn [2014], nous présentons quelques langages contrôlés, regroupés selon leur but premier, et leurs caractéristiques. Mais avant, nous présentons l'espace multidimensionnel de comparaison des langages contrôlés, toujours selon Kuhn [2014].

Langage naturel contrôlé

Un langage naturel contrôlé (LNC) parfois simplement appelé langage contrôlé, est un langage construit et qui s'appuie sur un langage naturel existant. Un LNC possède une grammaire, un vocabulaire, une expressivité plus limitée que le langage naturel support [Kuhn 2014], [Schwitter 2002]. Dans certains cas, il possède un style de manière à pouvoir le rendre complètement compréhensible par un ordinateur.

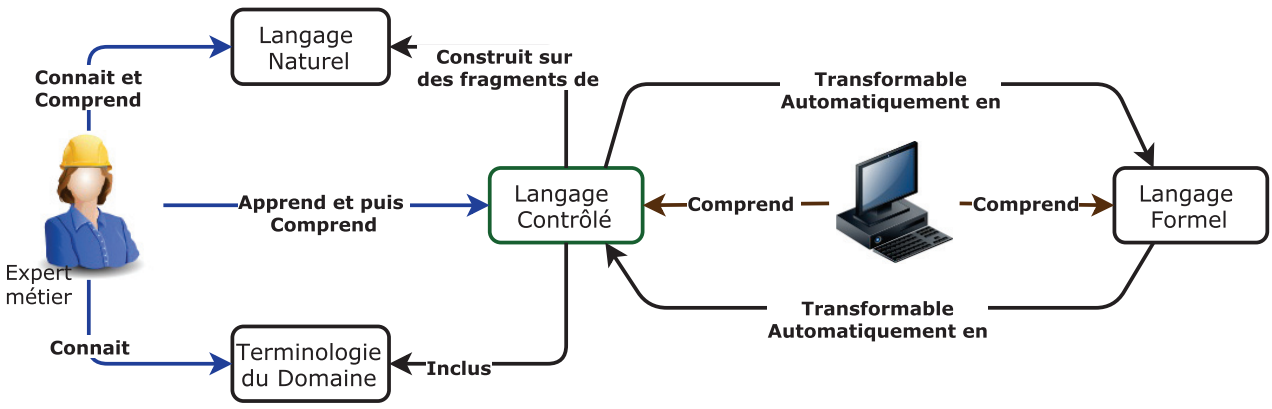


FIGURE 3.7 – Rôle pivot du langage contrôlé pour la représentation formelle des connaissances, par un non logicien.

3.4.2 Axes de comparaisons des langages naturels contrôlés

Types et propriétés des langages naturels contrôlés

En se basant sur différents travaux de la littérature ([Schwitter 2002], [Pool 2006], [Clark *et al.* 2009]), Kuhn [2014] fait ressortir neuf (09) grands types de langages contrôlés. Notons d'emblée qu'un même langage peut appartenir à plus d'un de ces types. Nous avons regroupé les types de langages contrôlés dans le tableau 3.2.

Tableau 3.2 – Lettres de code pour les types de langages naturels contrôlés. Basé sur [Kuhn 2014, p. 6].

Code	But du langage
C	Améliorer la ***Communication*** entre humains, principalement parlant des langues différentes
T	Améliorer les méthodes de ***traduction*** (semi-) automatiques
F	Faciliter une représentation ***formelle*** (et exécutable) du langage
W	Le langage est destiné à l'***écriture*** (*Writing*)
S	Le langage est destiné à l'***oral*** (*Spoken*)
D	Le langage est conçu pour un ***Domaine*** bien précis
A	Le langage est issu de travaux ***Académiques***
I	Des ***Industriels*** sont à l'origine du langage
G	Le langage est née d'une initiative ***Gouvernementale***

À ces types on pourrait en ajouter beaucoup d'autres. Par exemple le caractère *human-oriented* ou *machine-oriented* d'un langage, relevé par Huijsen [1998] (et repris dans d'autres classifications comme celle de Njonko Feuto [2014]). Pour cette distinction proposée par Huijsen, nous constatons avec Kuhn que les langages orientés humains sont compris dans les langages de types C (communication) et les langages orientés machines peuvent être rangés dans les catégories T (traduction) et F (formalisation).

Système de classification PENS

La typologie des langages contrôlés, telle que présentée ci-dessus concerne principalement les domaines d'application et la manière dont les langages seront utilisés. Par contre, les propriétés fondamentales, telles que présentées dans la littérature, n'apparaissent pas dans cette typologie. Par propriétés fondamentales, nous entendons l'***expressivité*** [Mitamura & Nyberg 1995, Pool 2006], la ***complexité*** [Mitamura & Nyberg 1995], les modifications grammaticales par rapport au langage naturel original [Pool 2006], la compréhensibilité, la non-ambiguïté et la prédictibilité [29], l'existence d'une définition formelle, l'aspect naturel du langage contrôlé [Wyner *et al.* 2009], etc. Nous voyons qu'il existe une multitude de dimensions pour explorer l'univers des langages contrôlés ; la définition de ces dimensions n'étant pas toujours claire avec parfois des dimensions qui se

29. C'est-à-dire "Est-ce que les phrases du langage contrôlés ont-elles une interprétation fixe dans le domaine d'application considéré ?"

recoupent. C'est le cas par exemple de la non-ambiguité + prédictibilité d'une part et l'existence d'une définition formelle d'autre part, telle que suggérée par Wyner *et al.* [2009]. Ainsi, pour réduire le nombre de dimensions et visualiser les langages contrôlés suivant des angles plus clairs, Kuhn [2010; 2014] propose un schéma de classification formé uniquement de quatre (4) axes qui sont : la ***Précision*** (*precision*), l'***Expressivité*** (*expressiveness*), la ***Naturalité*** (*naturalness*) et la ***Simplicité*** (*simplicity*). Ces quatre axes constituent le schéma ***PENS***. Chacun de ces axes possède cinq (5) niveaux qui vont de 1 (un) pour le plus faible à 5 (cinq) pour le niveau le plus haut.

1. **Précision.** Elle représente le degré avec lequel le sens d'une phrase en langage contrôlé peut être obtenu sans ambiguïté. Ses deux extrêmes sont le langage naturel (précision minimale) et le langage formel (précision maximale). Les cinq niveaux de précision sont :
   — Le niveau 1 (noté P^1) dit des langages ***imprécis***. Il contient les langages, tels que les langages naturels, qui ont besoin d'une intervention humaine pour décodage et qui peuvent donner des interprétations multiples d'une même phrase.
   — Les langages ***moins imprécis*** (P^2) dont l'interprétation, moins dépendante du contexte, est moins ambiguë que celle du niveau P^1. À ce niveau, on commence à observer les restrictions sur le langage naturel de base, ce qui permet de réduire les ambiguïtés.
   — Le niveau P^3 des langages ***à interprétation fiable*** qui, comme son nom l'indique, se compose de langages contrôlés dont une interprétation automatique est fiable. Ces langages ne disposent pas nécessairement d'une représentation formelle, et leur décodage repose sur des heuristiques ou des retours utilisateurs.
   — Les langages à ***interprétation déterministe*** (P^4) disposent d'une syntaxe formelle. Pour la grande majorité des phrases, on peut obtenir une unique représentation formelle mais il existe quelques cas rares où l'on hésite entre un nombre très réduit de représentations. Ainsi, il est parfois nécessaire de faire appel à des heuristiques ou à des ressources externes pour cerner complètement le sens d'une phrase.
   — Les langages dits à ***sémantique fixe*** constituent le niveau des langages les plus précis (P^5). À ce niveau, tout langage dispose d'une syntaxe et d'une sémantique formelle. Toute phrase possède un sens unique qui peut être décodé de manière automatique, sans recours à des heuristiques ou à des ressources externes.

2. **Expressivité.** Dans la classification PENS, l'expressivité d'un langage permet de circonscrire les types d'assertions qui peuvent être exprimés dans un langage. Un langage X est plus expressif qu'un langage Y si X permet de décrire toute chose descriptive en utilisant Y, mais pas l'inverse. Notons que, la relation "est plus expressif que" ainsi définie n'est pas transitive. Cela complique l'établissement

d'un classement linéaire des langages. Néanmoins, Kuhn propose un ensemble de cinq (5) types d'énoncés, sur lesquels on peut s'appuyer pour définir de manière rigoureuse des classes d'expressivité. Ces énoncés sont :

(a) La quantification universelle sur les individus,

(b) Les relations entre plusieurs entités (Par exemple : les relations binaires, ternaires, etc.),

(c) Les énoncés sous forme de règle (c'est-à-dire sous la forme ***si*** ... ***alors***),

(d) La négation,

(e) La quantification universelle de second ordre, sur les concepts et les relations.

Comme Kuhn le fait remarquer, on pourrait ajouter parmi ces types d'énoncés, les énoncés interrogatifs, déclaratifs, l'égalité, la quantification existentielle, etc. Mais les énoncés des types (a)-(e) ci-dessus, permettent de bâtir des niveaux d'expressivité pertinents pour explorer l'espace des langages contrôlés. De manière similaire à la dimension précision, PENS possède cinq (05) niveaux d'expressivité qui sont :

— Le niveau E^1 des langages dits ***inexpressifs*** qui ne permettent pas l'emploi des quantifications universelles - (a) - ***et/ou*** des relations entre plusieurs entités - (b).

— La classe E^2 des langages ***peu expressifs*** qui permettent des assertions basées sur les énoncés de types (a) et (b) ; mais ces langages ne sont pas de niveau E^3.

— Le niveau des langages à ***expressivité moyenne*** qui couvrent des assertions comportant des énoncés de types (a) - (d). Ainsi, à l'expressivité des langages E^2 on ajoute la possibilité d'utiliser la négation ou de s'appuyer sur le *logical pattern*. Bien sûr, les langages de ce niveau n'appartiennent pas au niveau suivant (E^4).

— Les langages à ***haut niveau d'expressivité*** (E^4) s'appuient sur tous les types d'énoncés (a) - (e). Toutefois, ils ne sont pas de niveau E^5.

— La classe des langages à ***expressivité maximale*** (E^5) qui permettent d'exprimer tous les types d'énoncés qui existent dans les communications humaines. Bien entendu, les langages de cette classe couvrent et vont au-delà des caractéristiques (a) - (e). C'est la classe des langages naturels.

3. **Naturalité.** Dans le schéma PENS, la naturalité permet de rapprocher la lisibilité et la compréhensibilité d'un langage contrôlé vis-à-vis du langage naturel support à ce langage contrôlé. On distinque ainsi :

— Les langages dits ***non naturels*** qui forment le plus bas niveau de naturalité (N^1). Comme leur nom l'indique, les langages de la catégorie N^1 ne semblent pas du tout naturels, emploient de nombreux caractères spéciaux ainsi qu'une pléthore de mots-clés. Ils laissent très peu de place aux expressions issues du langage naturel.

— Le niveau N^2 qui comprend les langages contrôlés avec une ***prédominance d'éléments non naturels***. Les langages de ce niveau intègrent des mots tirés du

langage naturel ainsi que des éléments non naturels. Néanmoins, la construction des phrases et leur sémantique sont assez éloignés de ce qu'on retrouve dans le langage naturel.

— Le niveau N^3 des langages contrôlés avec une ***prédominance d'éléments naturels*** qui emploient, entre autres éléments, des mots du langage naturel et dont les phrases possèdent une structure similaire à celle du langage naturel. Toutefois, l'utilisation d'éléments non naturels ne suffit pas à le considérer comme étant un langage naturel, malgré le fait que les personnes familières du langage naturel support puissent le comprendre.

— Les langages ***utilisant des phrases naturelles***, qui forment le niveau N^4. Ces langages utilisent des phrases naturelles et correctes des points de vue de la syntaxe et de la sémantique. Ils peuvent utiliser des codes de couleurs, des indentations, ou encore des tirets, des lettres majuscules ou minuscules, des parenthèses et autres délimiteurs qui entachent leur naturel. Ainsi, des textes ou documents écrits entièrement à partir de tels langages, ne paraîtraient pas du tout naturels.

— Les langages ayant la plus grande naturalité (N^5) sont appelés langages ayant des ***textes naturels***. Ici, ce ne sont pas seulement quelques phrases qui sont naturelles ou ont l'air naturel, mais bien l'ensemble du discours. Bien entendu c'est la classe par excellence des langages naturels avec toute leur diversité de styles.

4. **Simplicité.** Cette quatrième dimension du schéma PENS nous renseigne sur l'effort à fournir pour disposer d'une implémentation formelle de la syntaxe et de la sémantique du langage contrôlé. Notons que, ainsi définie, la simplicité ne concerne pas la facilité d'apprentissage du langage par les non initiés. Pour "quantifier" la notion de simplicité, Kuhn introduit un indicateur qui est le ***nombre de pages***[30], qu'il ***faudrait***[31] pour décrire le langage contrôlé en langage naturel, et ce de manière précise et compréhensible - c'est-à-dire fournir toutes les propriétés syntaxiques et sémantiques du langage, de manière rigoureuse (notations mathématiques si nécessaires) de sorte qu'un programmeur puisse développer un analyseur pour ce langage. Pour les langages dont le vocabulaire n'est pas fixe, ce dernier n'est pas pris en compte dans le schéma PENS, lors de l'estimation du nombre de pages pour la description du langage contrôlé. De manière analogue aux quatre dimensions précédentes, il existe cinq (5) niveaux de simplicité :

— Le niveau le plus bas, qui est occupé par les langages ***très complexes*** (S^1). Ils ont une complexité proche de celle des langages naturels, et ne peuvent être décrits d'une manière précise.

— Le niveau S^2 des langages ***sans description complète***. Ici les langages bien que proches du langage naturel, sont débarrassés des structures syntaxiques et sé-

30. Une page peut contenir jusqu'à 700 mots.
31. Une telle documentation peut exister ou non.

mantiques complexes. Toutefois, ils ne sont pas suffisamment simples pour être décrits de manière concise. Généralement, les descriptions de ces langages sont des listes de restrictions sur le langage naturel support.

— Le niveau des langages S^3 ayant une ***longue description***. Dans ce cas, la description existe ou peut exister mais sera contenue sur un très grand nombre de pages - plus de dix (10) pages.

— Les langages à ***description courte*** (S^4), différents de ceux de niveau S^3 par le nombres de pages - inférieur à 10.

— Les langages les plus simples, c'est-à-dire ayant une ***très courte description***. Ces langages peuvent être décrits de manière claire à l'aide d'une seule page.

Pour étoffer la description de la dimension S, Kuhn ajoutent que les langages des niveaux S^1 et S^2 sont construits sur la ***proscription*** de certains règles du langage naturel alors que les langages des trois autres niveaux, les règles de construction de phrases sont des ***prescriptions***, qui définissent pas à pas la manière dont les phrases doivent être écrites.

Avant de présenter quelques langages contrôlés proprement dit, rappelons que le schéma PENS (bien que disposant de plusieurs niveaux par dimensions), ou la typologie des langages contrôlés (voir tableau 3.2), ne servent pas à affirmer qu'un langage est meilleur ou pire qu'un autre. Ils nous permettent plutôt d'avoir une idée précise sur la nature des langages contrôlés. Si, sur un plan purement théorique, il est souhaitable de se situer au niveau le plus haut, c'est-à-dire le cinquième, sur chaque dimension du schéma PENS, en pratique et en fonction du domaine d'application d'autres souhaits peuvent être formulés.

3.4.3 Les Langages Naturels Contrôlés proprement dits

Dans son état de l'art et sa classification sur les langages naturels contrôlés, Kuhn [2014] identifie pas moins de cent (100) langages [32]. Ils couvrent une période allant de 1930 à 2012 [33]. Nous ne présenterons pas la totalité de ces langages ici. Seuls quelques-uns, répondant aux critères que nous énonçons ci-après.

À la figure 3.7, nous illustrions l'importance des langages contrôlés, en tant que pont entre langage-humain et langage-machine. Nous mentionnions le fait qu'ils garantissaient à l'expert métier une totale autonomie. Pour cela, les exigences réglementaires que ce dernier allait rédiger, devraient être ***transformables de manière automatique***, en des règles formelles. Par conséquent, seuls les langages de niveau P^5 au sens de PENS, c'est-à-dire ayant une précision maximale, présentent un intérêt pour notre travail. Ce premier filtre permet de réduire la liste des langages identifiés pas Kuhn de 100 à 34 langages. Parmi ces langages restants, nous nous attardons uniquement sur ceux

32. Une liste exhaustive de ces langages est disponible dans son état de l'art et récapitulée dans le fichier CSV suivant : `http://www.tkuhn.org/pub/cnlsurvey/data.csv`

33. Année de soumission d'une première version de son état de l'art.

qui ont été conçus pour répondre aux problématiques du Web Sémantique (comme la construction d'ontologies, la définition de règles d'inférences, etc.). Ce second filtre nous permet de conserver 17 langages naturels contrôlés. Enfin, parmi ces 17 langages, nous nous intéressons à ceux ayant un niveau d'expressivité supérieur ou égale à trois (3). En effet ce n'est qu'à partir de ce niveau qu'on couvre les énoncés sous forme de règles et l'utilisation possible de la négation. Ces deux types d'énoncés sont fondamentaux pour exprimer des exigences réglementaires. Nous obtenons ainsi 2 langages à la fois, complètement précis, orientés Web Sémantique et pouvant être utilisés pour exprimer des règles, à savoir : Processable English-D (***PENG-D***) et ***Pseudo Natural Language*** (PNL). Pour des besoins de simplification, nous regrouperons ces langages sous l'appellation ***"Langages naturels contrôlés orientés Web Sémantique"***. Certains de ces langages trouvent leur inspiration dans des langages plus généraux, ***Attempo Controlled English*** (ACE) et Processable English (***PENG***) notamment. Aux côtés de ces langages, nous présenterons quelques langages contrôlés "orientés règles" (mais pas de niveau P^5). Cela nous permettra de souligner les besoins à couvrir lorsqu'on envisage de mettre à disposition des experts métiers des langages contrôlés pour qu'ils rédigent des prescriptions. À la suite de ces deux groupes de langages contrôlés, nous présentons les initiatives qui ont été menées dans le domaine de la Construction, pour proposer aux experts métiers une représentation intermédiaire entre le langage naturel et les langages formels.

3.4.4 Langages Naturels Contrôlés Généraux

Attempo Controlled English (ACE)

ACE est un fragment de la langue anglaise, conçu pour fournir à des experts un moyen pour exprimer les connaissances métiers ; il se veut simple à apprendre, facile à lire et à écrire [Fuchs *et al.* 2006]. ACE est défini par des règles de constructions de phrases mais disposent aussi de règles d'interprétation qui servent d'heuristiques pour interpréter des expressions ambiguës (comme les références anaphoriques - pronoms relatifs, personnels, etc.). [Kaljurand 2007] et [De Coi *et al.* 2009] fournissent des mécanismes de correspondances directes entre un fragment de ACE et OWL 1.1, sans prise en compte des *datatype properties* (c'est-à-dire les propriétés qui permettent de relier des objets à des littéraux). Pour que ces correspondances demeurent simples et réversibles, elles ne prennent pas en considération les ***modificateurs*** tels que les adjectifs, les adverbes ou encore les propositions. Par ailleurs, notons que, bien que ce ne soit pas son but premier, ACE peut être utilisé pour écrire des règles [Kaljurand 2008, Kaljurand & Kuhn 2013]. Sur les référentiels fournis par Kuhn, le langage ACE est caractérisé par ($P^4E^3N^4S^3$, F W A) [Kuhn 2014, p. 18].

Processable English (PENG)

PENG est un langage naturel contrôlé proposé par Schwitter [2002]. Bien que très riche, une phrase en PENG peut être transformée automatiquement en une formule fidèle à la logique du premier ordre. C'est un langage inspiré d'ACE et développé pour écrire des spécifications pour des besoins de représentations des connaissances. Comme exemples de phrases écrites en PENG, on pourrait avoir (les termes en bleu sont les mots clés de PENG) :

(a) *Lifting-platforms and companies are objects.*

```
:Object rdf:type owl:Class.
:LiftingPlatform rdf:type owl:Class;
       rdfs:subClassOf :Object.
:Company rdf:type owl:Class;
       rdfs:subClassOf :Object.
```

(b) *For every lifting-platform there is a company that is the maker of the lifting-platform.*

```
:LiftingPlatform rdfs:subClassOf[
           rdf:type owl:Restriction;
           owl:onProperty :maker;
           owl:someValuesFrom :Company].
```

Si nous reprenons le schéma de IfcOWL présenté à la figure 3.5 et la règle en langage naturel "*Each building must have a storey fitted with fire doors.*", son écriture semi-formelle en utilisant PENG pourrait être[34] : ***For every ifc-building-storey that decomposes an ifc-rel-agregates that relating-object an ifc-building, the ifc-building-storey relating-structure an ifc-rel-contained-in-structure that related-elements an ifc-fire-door***. Nous voyons que la très bonne naturalité de PENG, N^4 sur le schéma PENS, est altérée par la non-naturalité des entités IFC telle que soulignée à la section 3.2.1 (page 38).
Utilisant le schéma PENS et la typologie des langages contrôlés, [Kuhn 2014, p. 40] caractérise PENG par ($P^5E^3N^4S^3$, F W A).

3.4.5 Langages Naturels Contrôlés Orientés Web Sémantique

PENG-D

[Schwitter & Tilbrook 2004] est un langage construit sur les bases de PENG. La principale différence est que PENG-D, moins expressif que PENG, est compatible avec les logiques de descriptions et implémenté en utilisant des fragments de RDFS et OWL. Ainsi Schwitter & Tilbrook [2004] font le choix de la décidabilité, en conservant un bon niveau d'expressivité.
Tout comme PENG, il est caractérisé par ($P^5E^3N^4S^3$, F W A).

34. La formulation n'est pas unique.

Metalog Pseudo Natural Language (PNL)

Metalog PNL [Marchiori 2004] a été conçu comme un langage convivial pour l'écriture d'assertions à destination du Web Sémantique. En tant que sur-couche de Metalog [35], il est construit sur RDF et la logique du premier ordre et utilise prolog pour effectuer les inférences. Illustrons l'approche Metalog avec la prescription enquote*Each building must have a storey fitted with fire doors.* et l'ontologie de la figure 3.5 :

1. Définition des "variables" (***toujours en majuscules***) :
```
BUILDING represents "http://example.com/LiftOnto#IfcBuilding".
STOREY represents "http://example.com/LiftOnto#IfcBuildingStorey".
DECOMPOSES represents "http://example.com/LiftOnto#decomposes".
AGGREGATE represents "http://example.com/LiftOnto#IfcRelAgregates".
ISRELATING represents "http://example.com/LiftOnto#relatingObject".
STRUCTURES represents "http://example.com/LiftOnto#relatingStructure".
RELATIONINSTRCTURE represents
"http://example.com/LiftOnto#IfcRelContainedInStructure".
ISRELATEDTO represents "http://example.com/LiftOnto#relatedElements".
FIREDOOR represents "http://example.com/LiftOnto#IfcFireDoor".
IS represents the verb "be" from the definitions in
"http://www.relationships.example.org/verbs".
NONCOMPLIANT represents "http://example.com/LiftOnto#NonCompliant".
```
2. Exigence proprement dite (***mots clés en minuscules***) : `if a STOREY DECOMPOSES an AGGREGATE that ISRELATING a BUILDING and that STOREY STRUCTURES a RELATIONINSTRCTURE that ISRELATEDTO a FIREDOOR then that STOREY IS NONCOMPLIANT.`

Au regard des spécifications de Metalog PNL, Kuhn [2014, p. 19] considère que ce langage a pour caractéristiques ($P^5E^3N^3S^3$, F W A).

3.4.6 Langages Naturels Contrôlés Orientés Règles Métier

Dans cette catégorie, nous avons les langages SBVR-SE, RULESPEAK, RuleCNL, RECON, Gherkin.

Semantics of Business Vocabulary and Business Rules - Structured English (SBVR-SE) et RULESPEAK

Pour les besoins des entreprises en matières de rédaction de spécifications, l'*Object Management Group* (OMG [36]) a publié un standard, SBVR. Ce standard fournit des mécanismes pour définir la syntaxe et la sémantique des règles, dans un style naturel, à destination des experts métiers. Toutefois, SBVR n'est pas à proprement parler un langage naturel contrôlé. En effet, SBVR ne dispose pas d'une syntaxe formelle, et il revient à chaque

35. Metalog étends RDF par des fonctions logiques (et, ou, l'implication), et des opérations arithmétiques.
36. `http://www.omg.org/`

personne s'appuyant sur les recommandations SBVR d'en proposer une [Spreeuwenberg & Healy 2009]. SBVR se concentre sur le sens des règles et ne dispose pas d'une syntaxe précise sur laquelle les experts doivent s'appuyer pour formuler ou reformuler leurs spécifications.

SBVR-SE et RULE SPEAK [Ross 2003b; 2009] sont deux langages naturels contrôlés pour l'écriture de règles. Tous les deux s'appuient sur la sémantique de SBVR. Bien qu'ils se rapprochent d'une syntaxe formelle, ce sont surtout des langages compilant de ***bonnes pratiques*** pour la rédaction de spécifications claires et concises. SBVR-SE et RULESPEAK s'appuient sur plusieurs expériences de rédactions de prescriptions mais restent informels [Kuhn 2010]. Cette absence de grammaire n'empêche pas que des phrases en SBVR-SE ou en RULESPEAK puissent être réécrites à l'aide du méta-modèle qui encapsule la sémantique de SBVR. À l'origine, aucun éditeur de règles n'était disponible pour SBVR-SE et RULESPEAK. Toutefois, pour faciliter la compréhension formelle de phrases écrites en utilisant ces deux langages, OMG [2008] et Ross [2009] proposent un ensemble de polices et de couleurs pour permettre de saisir la nature des éléments employés dans la phrase, comme l'illustrent les exemples de la figure 3.8 ci-dessous : l'exemple (a) semi-formalise l'exigence "*The rated speed of the lifting platform shall not be greater than 0,15 m/s.*" et l'exemple (b) la prescription "*Each building must have a storey fitted with fire doors.*". Pour SBVR-SE, nous reprenons, la remarque faite pour le langage PENG lorsqu'il faut écire des spécifications IFC-compatible, à savoir que les entités IFC dégradent l'aspect naturel du langage SBVR-SE.

En se référant aux propriétés des langages contrôlés et au schéma de classification PENS, Kuhn [2014, p. 43] caractérise SBVR-SE et RULESPEAK par ($P^3E^4N^4S^2$, C F W I).

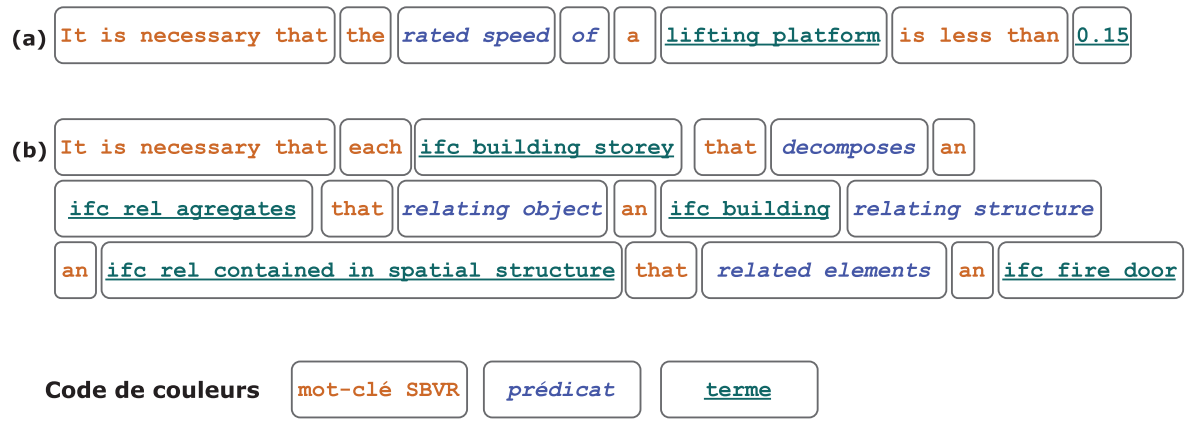


FIGURE 3.8 – Exemples de règles en SBVR-SE

RuleCNL

SBVR devenant de plus en plus populaire et avec lui SBVR-SE, des éditeurs du langage SBVR-SE ont vu le jour [Kamada *et al.* 2010]. Mais c'est seulement avec les travaux de Njonko Feuto *et al.* [2014] que l'on a pu avoir une syntaxe formelle pour SBVR. Ces

derniers proposent en plus un outil pour l'édition de règles en reprenant le formatage proposé par OMG [2008] pour SBVR-SE. Par l'existence de cette syntaxe formelle, RuleCNL est plus précis, au sens de PENS, que SBVR-SE et RULESPEAK. À la lecture de la description complète de RuleCNL [Njonko Feuto 2014, chap. 5], nous caractérisons RuleCNL par ($\mathbf{P^5}E^4N^4S^2$, C F W **A**). Les deux changements, sur la caractérisation de RuleCNL par rapport à ses prédécesseurs, portent sur :

— La précision, qui est passée du niveau P^3 au niveau P^5. RuleCNL est un langage à la précision maximale (P^5) car chaque phrase a une sémantique unique et est formalisée de manière automatique.

— L'origine de l'initiative du langage, qui est passée de I (industriel) à A (académique).

RECON - A Controlled English for Business Rules

RECON est un langage proposé par le National Institute of Standards and Technology (NIST) pour permettre à des experts, dans un domaine de connaissance donné, d'exprimer les connaissances de ce domaine. RECON est présenté dans un article de Barkmeyer & Neuhaus [2013]. RECON permet d'écrire des faits et des règles. Il nécessite de (re)définir les termes du domaine dans un dictionnaire selon un formalisme précis. Ces termes peuvent ensuite être utilisés pour écrire les informations du domaine. Il est important de préciser que ces termes doivent être des ***noms***, ***verbes*** et ***adjectifs*** du langage naturel support (c'est-à-dire l'anglais dans le cas de RECON). Cette condition marque une incompatibilité entre RECON et des ontologies de domaine ayant des termes aux libellés non-naturels (telles que IfcOWL). Notons toutefois que chaque phrase RECON a une sémantique précise et peut être automatiquement transformée en une expression formelle en *IKRIS Knowledge Language*[37] (IKL). RECON est un langage de coordonnées ($P^5E^3N^4S^3$, C F W I).

Gherkin

Lors du développement d'un logiciel, la rédaction des spécifications demeurent une partie cruciale ; ce sont elles qui décrivent ce qu'il faut exactement réaliser. Pour éviter les ambiguïtés à ce niveau et faire communiquer efficacement les parties prenantes et les développeurs, Wynne & Hellesoy [2012] proposent le langage contrôlé Gherkin pour la rédaction des spécifications. C'est un langage qui permet de décrire le comportement exact que doit avoir le futur logiciel face à des scénarios précis. Un exemple de spécifications en Gherkin est donné par le *listing* 3.4.

Listing 3.4 – Exemple de spécifications écrit en langage Gherkin

```
❶ #language: en
@basic
```

37. https ://www.ihmc.us/users/phayes/IKL/SPEC/SPEC.html

```
Feature:  Typification of entities

Background:
❷ Given  I have the ontology "IfcOWL"
And  I enrich it with the terms of the lifting platform glossary

Scenario Outline:   Succesful formalisation
❸ When  I enter the <Entity>
And  I submit the form
Then  The form should be redisplayed
❹ And  I should see <Type>

❺ Examples:
| Entity           | Type              |
| Lifting platform | Class             |
| rated speed      | Datatype property|
| 0.15             | float             |
```

Dans l'exemple ci-dessus, on voit que Gherkin :

— Permet de préciser la langue dans laquelle on écrit (ici `en` pour l'anglais - ligne ❶). Ceci a une incidence sur la langue des mots clés (de couleur bleue).

— Offre la possibilité d'utiliser des annotations personnalisées pour étiqueter des fonctionnalités[38] (en anglais *feature*) ; ces dernières étant décrites à l'aide d'un ensemble de scénarios (en anglais *scenario outline*)

À l'aide de cet exemple, précisons que derrière chaque ligne d'instruction (commençant par *Given*, *When*, *Then*, *But* ou *And*), Gherkin appellera une ou plusieurs fonctions du futur programme pour évaluer son comportement. Ces fonctions pourront avoir des ***paramètres*** qui en Gherkin sont soit des nombres, soient des dates, soit indiqués par des guillemets (par exemple `"IfcOWL"` - ligne ❷), soit précisés à l'aide de chevrons < et > (par exemple les valeurs des variables <Entity> et <Type> aux lignes ❸ et ❹ , sont fournies dans le tableau `Examples` en ligne ❺).

Kuhn [2014, p. 36] positionne Gherkin suivant les coordonnées ($P^5E^3N^4S^3$, F W D A).

Comme nous le verrons (cf. section 4.3.3), le langage contrôlé, RAINS, que nous proposons aux experts métiers pour la rédaction de règles semi-formelles, s'inspire de Gherkin. Ce choix vient du fait que Gherkin dispose des mécanismes d'annotations, d'identification simples de termes significatifs et l'isolation explicites de spécifications atomiques.

3.4.7 Langages dans le domaine de la Construction

À notre connaissance, il n'existe pas de langages naturels contrôlés spécifiques au domaine de la Construction. Cela vient notamment du fait que dans les travaux relatifs au contrôle automatique de la conformité, les règles formelles ont souvent été codées en

38. Idem pour les scénarios

dur. Par exemple, en SPARQL [Yurchyshyna *et al.* 2008a;b, Bouzidi *et al.* 2011; 2012], ou sous-forme d'axiomes en RDF/XML [Dimyadi & Amor 2013, Dimyadi *et al.* 2015], en N3Logic [Pauwels *et al.* 2011b;a], etc. Néanmoins, il existe des initiatives pour représenter des règles sous une forme graphique ou semi-formelle.

— Zhang *et al.* [2015] représentent des contraintes sous forme de ***patrons***, accompagnées de formules, si nécessaires, pour ajouter des précisions à ces patrons. Dans leurs travaux les patrons sont représentés sous une forme graphique respectant le formalisme du standard EXPRESS. L'exigence ainsi représentée ainsi que les formules éventuelles qui l'accompagnent sont représentées en mvdXML [39].

— [Solihin & Eastman 2015] proposent de représenter les règles à l'aide de ***graphes conceptuels***. Les graphes conceptuels ont été introduits par Sowa [1976], puis approfondi en 1984, et sont un formalisme graphique pour la représentation des connaissances et des raisonnements. Leur sémantique est calquée sur la logique du premier ordre.

Pour représenter les codes de Construction dans toute leur diversité, Solihin & Eastman enrichissent les graphes conceptuels par la possibilité de représenter la disjonction ("ou" logique), la négation, la capacité de spécifier des exceptions aux règles, etc.

— Le modèle **RASE** (*Requirement*, *Applicability*, *Selection*, *Exception*) [Hjelseth & Nisbet 2010; 2011] définit un ensemble de briques communes que l'on retrouve dans toute exigence réglementaire. Il mentionne que les codes de Construction contiennent un certain nombre de ***vérifications*** (en anglais *checks*), qui délimitent une portion distincte du code de Construction. En outre, chaque vérification est faite d'un certain nombre de ***contraintes à vérifier*** (en anglais *requirement*). De même, chaque vérification porte sur un ou plusieurs ***aspects*** (en anglais *applicability*). Éventuellement chaque vérification peut comporter des ***éléments sélectifs*** (en anglais *selection*) si l'exigence s'applique dans certains cas précis. Les vérifications peuvent aussi disposer des cas d'***exception*** (en anglais *exception*). L'exemple ci-dessous permet de mieux se rendre compte du découpage d'une contrainte réglementaire en langage naturel par l'une des quatre balises <r>, <a>, <s> ou <e> - chaque élément mis en exergue par une couleur bien précise est en réalité un morceau de texte compris entre une balise ouvrante et une balise fermante.

Le modèle RASE repose sur ces quatre types d'éléments, communs à toute exigence, pour permettre à des experts métiers ou des rédacteurs de normes de disposer de codes de constructions semi-formelle. Notons que le modèle RASE

39. http://www.buildingsmart-tech.org/specifications/mvd-overview

a été utilisé, avec succès, pour formaliser des codes de construction [Hjelseth & Nisbet 2011, Macit *et al.* 2015]. Toutefois, passer d'un texte annoté, comme celui de l'exemple ci-dessus, à des exigences formelles est ***manuel***. De plus, même si l'on suppose l'existence d'un modèle de données dont les termes sont directement alignés sur les termes du code de construction, ce qui est très rare en pratique, il faut encore franchir une difficulté pour passer d'annotations RASE à des expressions exécutables. En effet, en s'appuyant sur l'exemple ci-dessus, on peut penser que l'élément contraint est "la route d'accès" alors que pour être précis (c'est-à-dire se conformer au modèle de données), il pourrait s'agir plutôt de "la ***largeur*** de la route d'accès", voire de la "***largeur utile*** de la route d'accès", etc.

Synthèse

La revue de littérature sur les langages contrôlés que nous venons de présenter nous permet de nous rendre compte des faits suivants :

- À notre connaissance, il n'existe pas de langage naturel contrôlé, compatible avec la non-naturalité d'un langage comme les IFC (et donc avec le BIM).
- Parmi les langages contrôlés appropriés pour l'écriture des règles (SBVR-SE, RULESPEAK, RuleCNL), aucun ne possède une précision de niveau P^5 au sens de PENS. Autrement dit, la transformation de phrases écrites en utilisant ces langages, en expressions formelles, fait appel, dans certains cas, à des heuristiques. Or, comme nous le soulignons à l'aide de la figure 3.7, le besoin d'un langage de précision maximale est vital pour la confiance de l'expert métier. Rappelons que les exigences dans notre domaine d'application permettent d'attester de la conformité de plans de futurs bâtiments. Il faudrait par conséquent éviter des hypothèses sur la sécurité de ces derniers.
- PENG-D pourrait être le langage naturel contrôlé pivot pour l'écriture des codes de. Mais nous avons souligné que sa naturalité était altérée par les artifices présents dans les libellés des entités IFC et aussi par le recours aux relations objectifiées utilisées dans les IFC pour représenter les relations entre les objets.

Au chapitre 4, nous proposons un langage contrôlé, RAINS, qui adresse à ces limites.

3.5 Conclusion du Chapitre

Dans ce chapitre, nous avons présenté les principales notions que nous utiliserons pour bâtir nos contributions. Parmi ces notions, le concept de règles métiers, que nous appelons aussi prescriptions ou exigences réglementaires, occupe une place de choix. Nous avons présenté les différents angles sous lesquels la littérature explore la notion de règles ainsi que les défis à relever pour pouvoir disposer d'une expression formelle de ces dernières. Cela nous a permis de circonscrire le périmètre des règles métiers qui seront

prises en compte dans notre approche de formalisation. Ce périmètre comprend les règles strictes (obligations et interdictions) sous une forme textuelle (phrase), complètement interprétables.De plus ces règles devront être de simples inférences c'est-à-dire ne nécessitant pas d'action (voir section 3.1.5).
Une fois le concept de règles métiers présenté, nous nous sommes intéressés aux formalismes de représentation devant nous permettre de décrire nos règles ainsi que le monde qui les entoure. S'appuyant sur leurs qualités intrinsèques à savoir l'interopérabilité, la réutilisation et le partage des connaissances, les mécanismes d'inférences et la standardisation, nous avons choisi de travailler avec les langages du Web Sémantique (RDF, RDFS, OWL, SPARQL) pour la représentation des connaissances. Ce choix présente en plus l'avantage d'assurer la continuité avec les travaux antérieurs, relatifs au contrôle automatique de la conformité réglementaire de maquette de bâtiment.
Ces bases posées, nous avons fait une revue des approches (semi-) automatiques de formalisation des règles. Cela nous a permis de voir les principales hypothèses et les choix d'implémentation fait par nos prédécesseurs. Il ressort ainsi que l'existence d'un vocabulaire sémantisé des termes du domaine réglementé est primordial. Par ailleurs, ces travaux ont permis d'illustrer les difficultés que représentent l'alignement entre d'une part, les exigences réglementaires en langage naturel, et les termes du vocabulaire du domaine, d'autre part. À ces difficultés s'ajoutent, la constitution correcte des termes de l'antécédent et du conséquent de la règle. Dans notre travail, ce dernier a pour but premier de pouvoir affirmer si oui ou non un élément est conforme aux normes.
Enfin, nous avons clôturé ce chapitre sur les langages naturels contrôlés. Dans notre travail, ils nous servent de pont entre les experts métiers et la machine. Nous avons pu nous rendre compte qu'il existe plus d'une centaine de langages de ce type avec des syntaxes, des capacités, des buts aussi divers que variés. Par contre, lorsque nous nous sommes intéressés aux langages à la fois précis, compatibles avec les standards du Web Sémantique, et suffisamment expressifs (voir section 3.4.3), seuls deux langages se proposaient à nous : PENG-D et Metalog PNL. Nous avons souligné le fait que leur grande naturalité, qui contraste avec le caractère non naturel des termes des *Industry Foundation Classes*, était un frein pour une adoption dans le monde de la Construction. Nous avons aussi pu voir qu'il existe des langages naturels contrôlés non orientés Web Sémantique et dédiés à l'écriture de règles. Ce sont SBVR-SE, RULESPEAK, RuleCNL et RECON. Les deux premiers ne disposent pas d'une précision maximale. Ainsi, ils ne peuvent pas garantir à celui qui les utilise que les phrases qu'il écrit ne renferment aucune ambiguïté (voir page 70). Quant deux derniers, RECON et RuleCNL, ils perdent leur usage face à des vocabulaires renfermant des termes au libellés non-naturels. Nous avons terminé la présentation des langages contrôlés par le langage Gherkin. C'est un langage de rédaction de spécifications pour le développement d'applications. La présentation de Gherkin nous a permis de présenter le langage duquel nous nous sommes inspirés pour définir le nouveau langage RAINS. RAINS est le langage contrôlé que nous proposons. Il est compatible avec des vocabulaires non-naturels et adéquats pour réécrire les règles

métiers. En tant que langage naturel contrôlé adapté aux vocabulaires contenant des termes non-naturels, tel que le vocabulaire IFC, RAINS vient combler un vide pour la constitution de dépôts de règles métiers formelles.

Chapitre 4

RAINS : un langage contrôlé pour la formalisation de règles métiers

Contents

Le BIM (*Building Information Modeling*) et son standard de représentation des informations, les IFC (*Industry Foundation Classes*), connaissent une adoption croissante dans le secteur de la Construction. Ainsi, les bâtiments et autres installations disposent désormais d'une représentation numérique et standardisée. Ces maquettes numériques serviront à construire des ouvrages, qui en pratique doivent obéir à une multitude de normes. Avec l'existence d'une version numérique des futurs édifices, il devient possible de révéler, de manière automatique, les points de non-conformité sur les ouvrages dès la phase de conception. Cette opération nécessite, en plus de la maquette numérique des ouvrages, de disposer des normes sous une forme exécutable. Malheureusement, les prescriptions, rédigées en langage naturel, ne sont pas compréhensibles, en l'état, par les ordinateurs. Il est donc nécessaire de réécrire les exigences réglementaires sous une forme exécutable et compatible avec les standards de représentation de connaissances du BIM, à savoir les IFC.

Cette tâche de réécriture revient aux personnes ayant à la fois une connaissance du schéma des IFC et une connaissance métier du domaine de la Construction, qui leur permettent de comprendre les normes. Or, ces experts métiers ne disposent pas nécessairement d'acquis en logique et en représentation des connaissances. Une solution serait de permettre à ces experts de formuler les normes dans une représentation intermédiaire entre le langage naturel et les langages formels. Cette idée peut être implémentée à l'aide d'un ***langage naturel contrôlé***, tel que présentée à la section 3.4.3 et illustrée par la figure 3.7 (page 57).

Nous avons fait une revue de littérature des langages naturels contrôlés à la section 3.4. Cette étude des langages contrôlés existants nous a permis de conclure qu'il n'existait pas, à notre connaissance, de langage contrôlé à la fois :

1. Approprié pour des ontologies de domaines à faible naturalité ; ce qui est le cas du schéma des IFC ;
2. Adapté à l'écriture de règles,
3. De précision maximale, c'est-à-dire de niveau P^5 sur l'échelle de précision du schéma PENS (voir section 3.4.2),
4. Aligné sur les standards du Web Sémantique.

Cette conclusion nous conduit à proposer le langage naturel contrôlé ***RAINS***, qui réunit les conditions ci-dessus. Pour se faire, nous allons dans un premier temps présenter les critères auxquelles RAINS devra répondre (section 4.1). Ensuite, nous présenterons les types d'assertions que l'on peut retrouver dans une phrase RAINS (section 4.2). Ceci nous permettra de présenter la syntaxe formelle du langage RAINS (section 4.3), que nous illustrons à l'aide de quelques exemples (section 4.2), puis sa sémantique formelle (section 4.5). Les éléments du langage RAINS étant entièrement décrits, nous discuterons la position de RAINS dans l'espace formé par les dimensions du schéma PENS (section 4.6). Enfin, à l'aide d'exemples d'exigences réglementaires, nous étayerons la comparaison entre RAINS et PENG-D et Metalog PNL (section 4.7).

4.1 Approche de conception de RAINS

Dès les premiers moments de la conception du langage RAINS, nous avons voulu qu'il soit en phase avec le ***manifeste des règles métiers*** (*Business Rules Manifesto*) [Ross 2003a] et notamment avec son article 3. Dans les trois (03) premiers points de cet article, il est mentionné [1] :

1. "Les règles se fondent sur des faits, et les faits se fondent sur des concepts exprimés par des termes."
2. "Les termes représentent des concepts du domaine ; les faits sont des assertions relatives à ces concepts ; les règles contraignent et soutiennent ces faits."
3. "Les règles doivent être explicites. On ne doit pas être capable de contraindre un concept ou une assertion par une règle, sauf mention expresse."

Pour être en accord avec ces principes, nous avons souhaité que :

— Une exigence RAINS fasse explicitement mention du ***logical pattern*** $if\ p\ then\ q$;
— Chaque règle RAINS se compose d'un ensemble d'***assertions atomiques*** dont la distinction est tangible ;
— Les termes d'une exigence RAINS soient directement repris de l'ontologie conceptualisant le domaine et sont :
  — Soit des ***classes*** ou des ***individus*** de l'ontologie de domaine,
  — Soit des ***object*** et ***datatype properties*** ou des prédicats prédéfinis (par exemple des prédicats pour la comparaison - égal, supérieur, inférieur, etc.) pour fournir des informations sur les classes et les individus,
  — Soit des ***littéraux***,
  — Soit des ***marques de négation***,
  — Soit des ***marques de quantification/restriction universelle*** (par exemple : seulement, uniquement, etc.).

  Nous utiliserons l'expression ***entités RAINS*** pour désigner les termes pouvant être utilisés dans une règle RAINS, tels que listés ci-dessus.

Comme nous venons de le mentionner, une règle RAINS se compose d'assertions atomiques. Dans la section suivante, nous présentons les différents types d'assertions qu'on peut retrouver dans une règle RAINS.

4.2 Les assertions avec RAINS

Une phrase RAINS se forme sur la base d'assertions, chacune pouvant être d'un type disponible parmi les cinq (05) types d'assertions RAINS. Nous les présentons dans les paragraphes suivants. Bien que n'ayant pas encore introduit la syntaxe formelle de

1. Traduit de l'Anglais.

RAINS, nous allons illustrer chaque type d'assertion en la mettant en exergue dans une règle RAINS bien formée. Cependant, précisons que toute entité RAINS, c'est-à-dire un terme tel que décrit à la section précédente, est délimitée par une paire de ***guillemets dactylographiques*** ; c'est-à-dire le signe de ponctuation **"**. De plus, ***des nombres entiers sont utilisés pour matérialiser les co-références***.

4.2.1 *Single concept assertion*

C'est une assertion qui ne comporte qu'une entité RAINS qui fait référence à une classe de l'ontologie de domaine. Un exemple de *single concept assertion* est fourni dans le conséquent (partie *then*) de la règle RAINS ci-dessous (l'entité de couleur bleue représente le concept de la *single concept assertion*) :

```
If the "rated speed" of a "Lifting platform" is "greater than" "0.15" m/s
Then it is "Non-compliant"2
```

4.2.2 *Double concept assertion*

Comme son nom l'indique, une *double concept assertion* est une assertion qui se construit autour de deux entités RAINS qui sont des classes de l'ontologie de domaine. Ce type d'assertion permet de mentionner des concepts qui font référence l'un à l'autre et ne sont pas reliés à l'aide d'un prédicat. Pour illustration, nous avons l'antécédent (introduit par *When*) de la règle (les entités en bleu représentent les concepts de la *double concept assertion*) :

```
When a "Building storey" is "not" "Accessible to disable persons"
Then it is "Non-compliant"1
```

4.2.3 *Data assertion*

Une *data assertion* permet de préciser la valeur de l'attribut d'un concept. Par ailleurs, cet attribut fait référence à une *datatype property* de l'ontologie de domaine. Par exemple, l'antécédent (introduit par *If*) de la règle ci-dessous est une *data assertion* (l'entité de couleur bleue représente l'attribut c'est-à-dire la *datatype property*) :

```
If a "Lifting platform" has a "rated speed" that is "greater than" "0.15" m/s
Then it is "Non-compliant"1
```

Comme cela se produit dans les assertions en langage naturel, l'attribut peut venir avant le concept qu'il permet de caractériser. Par exemple :

```
If the "rated speed" of a "Lifting platform" "is greater than" "0.15" m/s.
Then it is "Non-compliant"2
```

4.2.4 *Boolean assertion*

On se sert d'une *boolean assertion* pour préciser l'état d'un concept à l'aide d'un *datatype property* dont la valeur peut être soit "*true*" ("vrai") ou "*false*" ("faux"), autrement dit un attribut booléen. Pour illustration, l'antécédent de la règle ci-dessous est une *boolean assertion* (l'entité de couleur bleue représente l'attribut booléen) :

```
If a "Lifting platform" "does not" have an "emergy control device"
Then it is "Non-compliant"1
```

Comme dans le cas d'une *data assertion*, il peut arriver que l'attribut booléen soit mentionné avant le concept qu'il permet de décrire. Par exemple :

```
If an "emergy control device" "is not" found on a "Lifting platform"
Then this "Lifting platform"3 is "Non-compliant"
```

4.2.5 *Object assertion*

Une *object assertion* est une relation binaire entre deux concepts ou entre un concept et un individu. Elle est supportée à l'aide d'une *object property*. Elle est construite sur le même modèle qu'une *data assertion*. La différence réside en l'utilisation d'une *object property* à la place d'une *datatype property*. Un exemple d'*object assertion* est donné par une assertion de l'antécédent de la règle RAINS ci-après (l'entité de couleur bleue représente l'*object property*) :

```
Given an "Ifc door type" that "has property sets" an "Ifc door panel properties"
If this "Ifc door panel properties"3 has a "panel width" that is "less than"
"80" cm
Then This "Ifc door type"1 is "Non-compliant"
```

Une fois de plus, il peut arriver que dans certaines *object assertions*, le prédicat précède le concept qu'il permet de décrire. Pour illustration (le prédicat inversé est écrit en bleu) :

```
Given an "Ifc door type" that "has property sets" an "Ifc door panel properties"
If the "panel operation" of this "Ifc door panel properties"3 is "not" "equal
to" "DOUBLE_SWING"
Then This "Ifc door type"1 is "Non-compliant"
```

Nous venons de présenter les cinq types d'assertions pouvant servir à écrire une règle RAINS. Ces catégories d'assertions se basent sur la manière dont les ***faits atomiques*** sont énoncés en langage naturel[2]. Par faits atomiques, nous entendons (*i*) des faits desquels on ne peut extraire d'autres faits et (*ii*) dont la suppression d'une entité (hors marque de négation ou de quantification universelle) présente dans ces faits, conduirait à une information partielle, voire un non sens. Par exemple :

2. Pour des langues comme le Français ou encore l'Anglais.

— “La vitesse nominale de la plateforme doit être supérieure à 0.15 m/s et inférieure à 0.25 m/s” n’est pas atomique car on peut y extraire les faits atomiques (1) “La vitesse nominale de la plateforme doit être supérieure à 0.15 m/s” et (2) “La vitesse nominale de la plateforme doit être inférieure à 0.25 m/s” ;
— “La "vitesse nominale" de la "plateforme" doit être "inférieure à" "0.25" m/s” est ***atomique***, car en supprimant une des entités (entre guillemets) , l’information délivrée par cette assertion serait erronée ou incomplète et inexploitable.

Pour illustrer les types d’assertions proposées par RAINS, nous nous sommes appuyés sur des exemples de règles RAINS valides, laissant entrevoir la syntaxe de RAINS. Dans la section suivante nous présentons la syntaxe formelle du langage contrôlé RAINS.

4.3 Syntaxe du langage RAINS

Un des objectifs de RAINS est d’être de niveau de précision P^5 sur l’échelle PENS. Autrement dit, RAINS doit disposer d’une syntaxe formelle et qui ne laisse place à aucune ambiguité. Nous avons formalisé la syntaxe de RAINS en utilisant la notation EBNF (*Extended Backus-Naur Form*) [Scowen 1998]. EBNF est une notation pour la spécification de grammaire de type hors contexte. En d’autres termes, des grammaires dans lesquelles les règles de dérivations sont de la forme

$$X \rightarrow \alpha$$

où X est un symbole non-terminal et α une expression comprenant des symboles terminaux et/ou non-terminaux. Le tableau 4.1 présente les symboles de la notation EBNF utilisés pour décrire la syntaxe de RAINS.

4.3.1 La syntaxe de RAINS proprement dite

La grammaire de RAINS proprement dite est décrite dans le *listing* 4.1. L’usage des couleurs sert uniquement à des besoins de lisibilité : une règle RAINS est en texte simple (c’est-à-dire *plain text*). Dans ce *listing*, on peut se rendre compte que :

— Le langage RAINS permet d’accompagner une règle de zéro ou plusieurs ***annotations***. Comme nous le verrons dans la section 4.4 dédiée à des exemples de règles RAINS, une annotation permet de définir le contexte dans lequel une règle s’applique.
— Une règle RAINS, proprement dite, se compose d’un ***antécédent*** et d’un ***conséquent*** qui sont rendus explicites par la présence obligatoire du terme `Then` dans toute règle.
— L’antécédent est constitué d’un ensemble de lignes. Chaque ligne débute par un mot clé pris dans la liste `If`, `Given`, `When`, `And`, `But` et se poursuit par une assertion

Tableau 4.1 – Symboles de la notation EBNF et leur description

Symboles	Signification
`mot hors ' '` (*unquoted words*)	Symbole non terminal
'...'	Symbole terminal
\d	Chiffre
+	Une ou plusieurs fois l'expression précédente
*	Zéro ou plusieurs fois l'expression précédente
?	Zéro ou une occurrence de l'expression précédente
... `Description` ...	`Description` textuelle d'un symbole
=	Définition de la règle
,	Concaténation
|	Alternatives (ou)
;	Fin de la règle
(*... *)	Commentaires

atomique.

— On peut exprimer le conséquent à l'aide d'une ou plusieurs lignes. Cependant, la première de ces lignes est introduite par le mot-clé `Then` , et les autres lignes, s'il y en a, par le mot clé `And` . En plus du mot clé, le reste de la ligne est une assertion atomique.

— Comme nous l'avons fait remarquer avec l'ontologie IfcOWL (voir figure 3.5, page 38 et suivantes), parfois, un prédicat en langage naturel se traduit par une chaîne de propriété en langage formel. RAINS permet de représenter cette chaîne de propriétés en les concaténant à l'aide d'un ***point*** ('.') c'est-à-dire sous la forme `"propriété1"."propriété2"...`

— Dans une règle RAINS, pour signaler qu'un concept D fait référence à un concept C, déjà mentionné dans cette règle, on rattache à D un entier n. Cet entier n, accolé à D permet de signaler que D fait référence à la $n^{ème}$ entité de la règle, qui est l'entité C.

— Certaines ontologies disposant d'entités dont les libellés peuvent être ***non naturels***, c'est le cas de IfcOWL, RAINS permet d'utiliser librement des expressions en langage naturel pour lier les termes et ainsi accroitre la naturalité des règles en langage contrôlé. Ces morceaux d'expression en langage naturel sont symbolisés par `Text`.

Listing 4.1 – Extrait de la syntaxe EBNF de RAINS.

```
Rule = Annotation*, Antecedent, Consequent;
Annotation = '@', AnnotationName, '(', (AnnotationValue, ','?)+, ')';
Antecedent = StartingKeyword, Assertion, ((StartingKeyword|'And'|'But'),Assertion)*;
StartingKeyword = 'If'|'Given'|'When';
Consequent = 'Then', Assertion, ('And', Assertion)*;
```

```
Assertion = ( SingleConceptAssertion | DoubleConceptAssertion
        | DataAssertion | BooleanAssertion | ObjectAssertion ), CarRetrn;

SingleConceptAssertion = Text?, Concept, Text?;
DoubleConceptAssertion = Text?, Concept, Text?, Neg?, Text?, SimpleConcept, Text?;

BooleanAssertion = OrderedBooleanAssertion | InvertedBooleanAssertion;
OrderedBooleanAssertion = Text?, Concept, Text?, Neg?, Text?,
 ChainOfObjectProperties?, BooleanDatatypeProperty, Text?;
InvertedBooleanAssertion = Text?,  BooleanDatatypeProperty,
 Text?, Neg?, Text?, Concept, Text?;

DataAssertion = OrderedDataAssertion | InvertedDataAssertion;
OrderedDataAssertion = Text?, Concept, Text?, Neg?, Text?, ChainOfObjectProperties?,
                DatatypeProperty, Text?, Neg?, Text?, Comparator, Quantity, Text?;
InvertedDataAssertion = Text?, DatatypeProperty, Text?, Concept, Text?,
                                        Neg?, Comparator, Quantity, Text?;

ObjectAssertion = OrderedObjectAssertion | InvertedObjectAssertion |
OrderedObjectAssertionWithIndiv | InvertedObjectAssertionWithIndiv;

OrderedObjectAssertion = SimpleObjectAssertion | Text?, Concept, Text?, Neg?, Text?,
ChainOfObjectProperties, (UniversalRestrictionKeyword | Comparator, Integer)?,
                 Text?, (Concept | ConceptWithAddon), Text?;
SimpleObjectAssertion = Text?, Concept, Text?, Neg?, Text?, ChainOfObjectProperties,
          Text?, Concept, Text?;
InvertedObjectAssertion = Text?, ObjectProperty, Text?, Concept, Text?, Neg?, Text?,
    (Concept | ConceptWithAddon), Text?;
ConceptWithAddon = SimpleObjectAssertion | OrderedDataAssertion;

OrderedObjectAssertionWithIndiv = Text?, Concept, Neg?, ChainOfObjectProperties,
                 Text?, Individual, Text?;
InvertedObjectAssertionWithIndiv = Text?, Individual, Text?, Neg?, Text?,
                 ChainOfObjectProperties, Concept, Text? |
        Text?, ObjectProperty, Text?, Concept, Text?, Neg?, Individual, Text?;

❶ ChainOfObjectProperties = (ObjectProperty,'.'?)+;

Quantity = Literal | Text?, DatatypeProperty, Text?, Concept;
❷ Concept = SimpleConcept, Coreference?;
SimpleConcept = '"', LabelOfDomainConcept ,'"';
Coreference = Integer;
Individual = '"', LabelOfDomainIndividual,'"';
DatatypeProrperty = '"', LabelOfDomainDatatypeProrperty,'"';
ObjectProrperty = '"', LabelOfDomainObjectProrperty,'"';
BooleanDatatypeProrperty = '"', LabelOfDomainBooleanDatatypeProrperty,'"';
❸ Literal = Integer | Float | String | Date;
Integer = \d+;   (* And also expressions for Float, String, Boolean and Date *)
Neg = '"no"'|'"not"'|'"is not"'|'"do not"'|'"does not"'; (* negation *)
```

```
UniversalRestrictionKeyword = '"only"'|'"exclusively"'|'"uniquely"';
Comparator = Equal | GreaterThan | GreaterThanOrEq | LessThan | LessThanOrEq;
Equal = '"is"'|'"is equal to"'|'"equal to"'|'"equal"'|'"equals"';
GreaterThan = '"more than"'|'"greater than"'|'"is greater than"'|'"exceed"'|';
(* And also expressions for GreaterThanOrEq, LessThan and LessThanOrEq *)
Text = ...free text in natural language... ;
CarRetrn = ...the carriage return char... ;
```

4.3.2 La syntaxe des entités RAINS

Une phrase RAINS reposent sur les entités RAINS. Nous les décrivons dans les items ci-après.

- Les termes du domaine (issus de l'ontologie du domaine), les prédicats prédéfinis, les litéraux et les marques de négations, sont des entités RAINS, et sont signalés par une paire de guillemets dactylographiques (").
- Dans le *listing* 4.1, les symboles non terminaux présents dans les assertions et colorés en rouge, à savoir `Concept`, `DatatypeProperty`, `ObjectProperty`, `Individual` et `BooleanDatatypeProperty` , font référence à des termes de l'ontologie de domaine qui sont respectivement : une ***classe***, une ***datatype property***, une ***object property***, un ***individu*** et une ***datatype property*** dont le ***range*** (`rdfs:range`) est de type booléen.
- Parmi les entités RAINS :
  - Les concepts, définis par le symbole `Concept`, commencent par une ***lettre majuscule***. Ils peuvent être suivis par un entier pour signifier une co-référence (voir la ligne ❷ du *listing* 4.1) ;
  - Les propriétés, qui sont référencées par `DatatypeProperty`, `ObjectProperty` et `BooleanDatatypeProperty`, commencent par une ***lettre minuscule*** ;
  - Les marques de négation, définies par le symbole `Neg`, les restrictions universelles, définies par le symbole `UniversalRestrictionKeyword`, ainsi que les prédicats de comparaison, définis par `Comparator`, commencent par une ***lettre minuscule***.
  - Les individus, définis par `Individual`, sont ***entièrement écrits en lettres majuscules*** et peuvent contenir des espaces et des *underscores*.

Pour conclure la présentation de la syntaxe de RAINS, nous disons que : ***une phrase RAINS se présente comme une* règle *(antécédent et conséquent explicites) dans un langage contrôlé, dans laquelle les entités formelles sont* annotées *(à l'aide de guillemets) et précédées éventuellement d'éléments spécifiant le contexte d'application de cette règle***.

Nous avons mentionné au terme de la section 3.4.6 (page 65) que le langage RAINS s'inpire de Gherkin. Nous présentons les points de ressemblance et de dissemblance entre les deux langages.

4.3.3 Rapprochement entre Gherkin et RAINS

Ce que RAINS doit à Gherkin

Certains choix dans la syntaxe de RAINS sont inspirés du langage Gherkin [Wynne & Hellesoy 2012] (voir *listing* 3.4 pour un exemple de spécifications Gherkin). Parmi ces choix on retrouve :

— La délimitation des entités à l'aide des guillemets,
— L'écriture des assertions chacune sur une ligne propre,
— L'introduction des assertions par une série de mots-clés donnés (`If`, `Then`, etc.),
— La possibilité d'annoter les règles RAINS.

Spécificités de RAINS

Si la syntaxe de RAINS s'inspire de certains aspects de Gherkin, il existe des différences entre les syntaxes des deux langages. Au rang de celles-ci :

— L'existence d'un vocabulaire de domaine. En effet, RAINS permet d'écrire des règles métiers et suppose donc l'existence d'un vocabulaire décrivant le domaine encadré par ces règles et chaque règle RAINS doit mentionner des termes de ce vocabulaire. Dans RAINS, c'est les termes de l'ontologie de domaine qui constituent le vocabulaire.
— La restriction de la syntaxe des assertions. Chaque assertion en langage RAINS doit correspondre à l'un des patrons des différents types d'assertions RAINS.
— L'utilisation d'une syntaxe plus élaborée pour les annotations. Dans RAINS, la définition des annotations n'est pas libre. En effet, chaque annotation RAINS s'appuie sur les termes de l'ontologie de domaine et a une sémantique encadrée par le schéma de ladite ontologie (cf. section 4.5.1).
— La possibilité d'indiquer des co-références.
— La possibilité d'utiliser des agrégats d'entités (chaines de propriétés).
— La délimitation de toutes les entités RAINS à l'aide de guillemets (dans Gherkin les littéraux ne sont pas explicitement signalés).

En plus de présenter des différences syntaxiques entre RAINS et Gherkin, le sens des phrases de chacun de ces langages n'est pas du tout le même. Par exemple, chaque phrase RAINS est un tout qui dénote une inférence et le sens de chaque assertion est défini par une ontologie de domaine. Or, dans le cas de Gherkin, un texte présente un ensemble de spécifications qui peuvent être prises individuellement et ces spécifications ne sont pas des inférences et n'ont pas de sémantique précise. En effet, bien qu'une spécification Gherkin soit analysée (en anglais *parse*) automatiquement, elle n'a de sens que pour le lecteur humain duquel on ***espère*** une compréhension claire de la spécification ; ce dernier devant ensuite implémenter cette spécification selon sa compréhension.

La syntaxe du langage RAINS étant présentée, nous allons nous familiariser avec ce

langage au travers de quelques exemples.

4.4 Exemples de règles RAINS

Dans cette section nous présentons quelques exemples de règles RAINS. Ces règles supposent l'existence d'un vocabulaire représenté par l'ontologie de domaine de la figure 4.1. Cette ontologie reprend le schéma IfcOWL représenté à la figure 3.5 et y ajoute quelques entités relatives aux plateformes élévatrices (`LiftingPlaform`, `ratedSpeed`, `length`), ainsi que la classe, `NonCompliant`, des objets non-conformes à la réglementation.

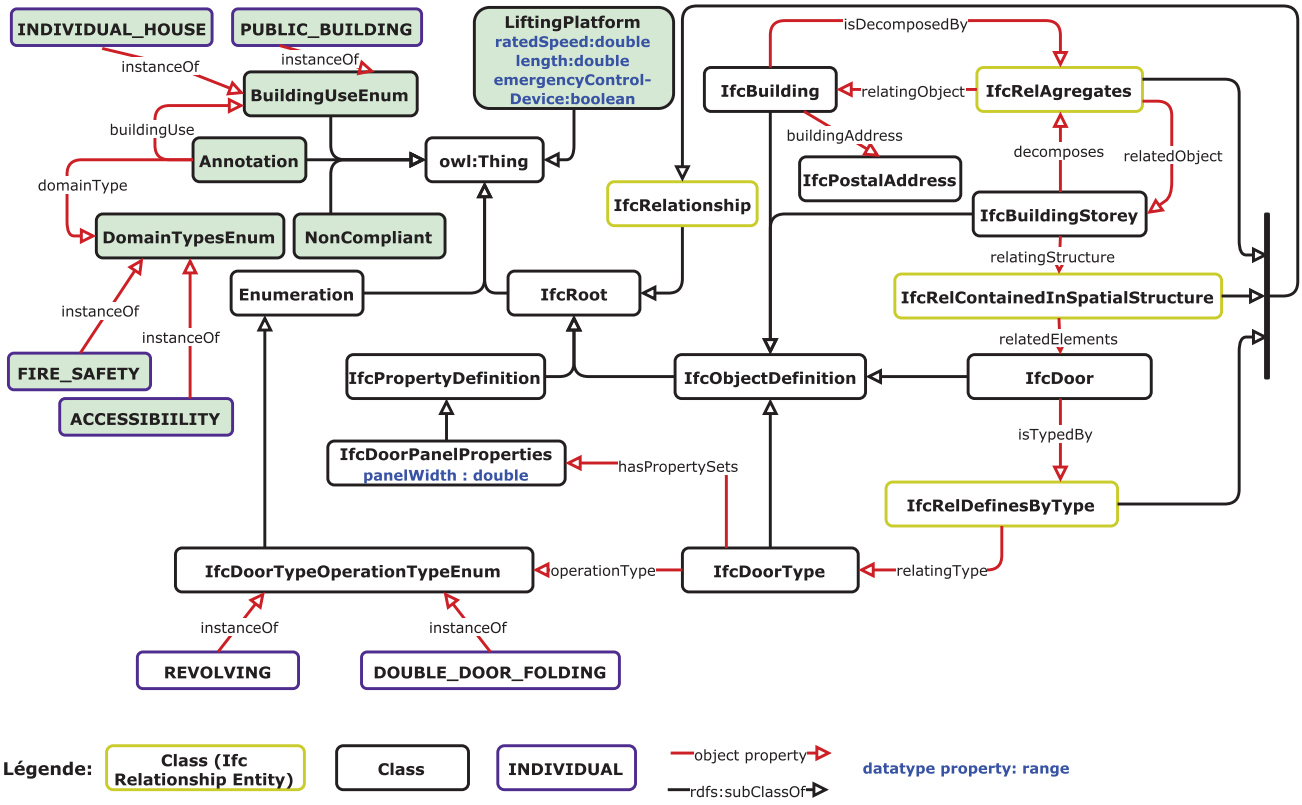


FIGURE 4.1 – Extrait de l'ontologie IfcOWL [Pauwels & Terkaj 2016] enrichie d'entités relatives aux plateformes élevatrices et d'entités servant à annoter les exigences. Ces nouvelles entités sont signalées par un fond de couleur verte.

Exemple 1 : *"The rated speed of the lifting platform shall not be greater than 0,15 m/s"*

Cette règle se traduit en français par "La vitesse nominale de la plateforme élévatrice ne doit pas dépasser 0.15 m/s". Nous pouvons constater que cette exigence n'est pas sous la forme ***if ... then***. La syntaxe de RAINS, obligeant à expliciter l'antécédent et

le conséquent de chaque exigence, la règle ci-dessus peut être reformulée de l'une des manières suivantes[3] : (1) *"If the rated speed of a lifting platform is not less than 0.15 m/s then it is non-compliant"* ou encore (2) *"If a lifting platform's rated speed is not less than 0.15 m/s then it is non-compliant."*, etc. L'une de ces deux formulations peut servir à réécrire l'exigence originale selon la syntaxe RAINS, qui pourrait être :

```
If the "rated speed" of a "Lifting platform" is "not" "less than or equal to" "0.15" m/s
Then it is "Non-compliant"2
```

Cette règle RAINS copie le schéma indiqué par la reformulation (1) de la règle originale. Son antécédent est une *inverted data assertion* introduit par le mot clé `If` . Par contre son conséquent est une *single concept assertion*. Par ailleurs, l'entité RAINS du conséquent, `"Non-compliant"2`, est un concept portant une marque de co-référence, à savoir l'entier 2. Ce chiffre indique que, le concept `"Non-compliant"` fait référence à ***la deuxième entité RAINS de la règle***, à savoir `"Lifting platform"` (la première entité RAINS étant `"rated speed"`). Au regard de cette exigence réglementaire, c'est donc l'objet *lifting platform* qui sera marqué du sceau de non-conformité (si sa vitesse nominale dépasse 0.15 m/s).

Par ailleurs, remarquons l'importance de l'utilisation libre du langage naturel entre des assertions RAINS. En plus de laisser le choix au rédacteur de la règle d'utiliser les articles, les propositions, les mots qu'il souhaite pour rendre la règle lisible et compréhensible, on se rend compte qu'il peut, dans le cas de valeurs numériques par exemple, préciser l'unité dans laquelle la valeur est donnée. La vitesse aurait pu être en `mètre par seconde` et écrite ***m/s***, $m.s^{-1}$, ***mètre par seconde*** directement ou en ***kilomètres par heure*** et écrite ***km/h***, $km.h^{-1}$, etc.

Notons que si l'on s'était aligné sur la reformulation (2) de la règle, alors la règle RAINS pourrait être :

```
Given a "Lifting platform" having a "rated speed" that is "not" "less than or equal to" "0.15" m/s
Then it is "Non-compliant"1
```

On constate que, cette fois ci, l'antécédent est une *ordered data assertion* et que la référence de l'entité `"Non-compliant"` pointe sur la première entité qui demeure `"Lifting platform"`.

Remarques :

1. Chacun des exemples que nous proposons est validé par les experts du métier.
2. La formulation d'une exigence selon la syntaxe de RAINS ***n'est pas unique***. C'est la raison pour laquelle nous utilisons des expressions comme "alors la règle RAINS pourrait être" pour introduire ***une*** expression RAINS pour une règle donnée. En effet, on peut choisir :

3. On peut reformuler cette exigence d'une multitude de façons différentes. L'important est de rester fidèle au schéma *if . . . then* et de conserver strictement la prescription de la règle originale.

— D'introduire les assertions par des mots différents (`If` , `Given` , `When` , etc.) ;
— D'inverser la position du concept par rapport à la propriété qui le décrit ;
— D'utiliser des chaines de propriétés ou non (voir exemple 3) ;
— D'utiliser chacun à sa manière, la liberté qu'offre RAINS de faire usage de texte en langage naturel pour relier les entités ;
— De délimiter différemment les prédicats prédéfinis (dans le *listing* 4.1, on voit par exemple que, RAINS considère valable les symboles terminaux `"is greater than"` , `"greater than"`, tous les deux faisant référence à la supériorité stricte).

Exemple 2 : "*In buildings with public access, the platform length shall not be less than 1 400 mm, to enable sufficient space for an attendant.*"

Comme dans l'exemple précédent, cette exigence ne met pas en avant de manière explicite la forme *If ... then*. Elle doit être reformulée pour respecter la syntaxe de RAINS. On pourrait la reformuler ainsi "*In buildings with public access, if the platform length is not greater than or equal to 1400 mm, then it is non-compliant*". En réécrivant cette phrase de manière compatible avec RAINS, on pourrait avoir :

```
@ Building Use(Public building)
Given a "Lifting platform" with a "length" that is "not" "greater than or equal
to" "1400" mm
Then it is "Non-compliant"1
```

La nouveauté par rapport à l'exemple précédent est la présence d'une annotation, à savoir `@ Building Use(Public building)`. Cette annotation précise que : (*i*) Dans le corps de la règle, telle qu'écrite en langage naturel, il est mentionné l' ***usage du bâtiment*** (*building use*) concerné par cette règle. (*ii*) Les types de bâtiments concernés par la règle sont les ***Établissements recevant du public*** (*public building*). Nous voyons qu'une annotation ne fait partie ni de l'antécédent ni du conséquent de la règle. Ainsi, elle n'est pas utile lors de l'exécution de la règle proprement dite. Cependant, une annotation trouve son utilité dans les systèmes de gestion de règles. Comme son nom l'indique, elle annote, décrit une règle. Dans un dépôt de règles, lors du contrôle de conformité d'un type de bâtiment précis, on peut par exemple s'intéresser uniquement aux règles concernant les établissements recevant du public, ou aux maisons individuelles, etc. Grâce aux annotations, on peut sélectionner les règles correspondant à un cas d'utilisation donné. Pour plus d'informations sur l'importance et l'utilisation des annotations, se référer à [Yurchyshyna 2009, chap. 6].

À la section 3.1.3 en page 27, nous avons effectué le constat selon lequel certaines propositions d'une phrase en langage naturel étaient inutiles à sa formalisation. Cela a donné lieu à ce que nous avons appelé ***élagage*** et qui est l'un des défis à relever pour la formalisation des normes. Ces propositions non pertinentes pour la formalisation le sont tout autant à la semi-formalisation que représente la transformation d'une exigence en une

règle RAINS. C'est le cas dans cet exemple de la proposition *"to enable sufficient space for an attendant"*. Elle permet de justifier la valeur minimale (1400 mm) fixée pour la longueur de la plateforme élévatrice, mais elle ne fournit aucune information pertinente pour permettre de relever des défauts sur une plateforme.

Exemple 3 : *"For doors with multiple panels, one of the panels must have a minimal width of 0.80 meter."*

Pour cette règle aussi, il est nécessaire de la repenser sous une forme explicite de règle. Cela peut conduire à : *"If a door has multiple panels and none of its panels' width is not at least 0.80 meter, then this door is non-compliant"*. En se basant sur cette reformulation on pourrait avoir les règles RAINS suivantes :

(a) ① If an "Ifc door" "is typed by" an "Ifc rel defines by type"
② And this "Ifc rel defines by type"3 "relating type" an "Ifc Door type"
And the "operation type" of that "Ifc door type" "is" "DOUBLE_DOOR_FOLDING"
③ And this "Ifc door type"6 "has property sets" an "Ifc door panel properties"
④ And this "Ifc door panel properties"9 has a "panel width" that is "not" "at least" "800" mm
Then this "Ifc Door"1 is "Non-compliant"

(b) ❶ If an "Ifc door" "is typed by"."relating type" an "Ifc door type"
And the "operation type" of that "Ifc door type"4 "is" "DOUBLE_DOOR_FOLDING"
❷ And this "Ifc door type"4 "has property sets"."panel width" that is "not" "at least" "800" mm
Then this "Ifc Door"1 is "Non-compliant"

Les phrases RAINS en (a) et (b) sont équivalentes. Bien qu'elles n'utilisent pas le même nombre d'assertions, et font appel à des entités différentes, elles décrivent la même réalité. La différence entre ces deux phrases est uniquement syntaxique. Elle réside dans le fait que la phrase (b) prend avantage du mécanisme de ***chainage de propriétés*** qu'offre la syntaxe de RAINS, contrairement à la phrase (a). En effet, on remarque que :

— Les assertions ① et ② de la règle RAINS (a) sont équivalentes à l'assertion ❶ de la règle RAINS (b). Cette dernière mentionne que, partant du concept "Ifc door", il est nécessaire d'emprunter le chemin composé des propriétés "is typed by" et "relating type" pour atteindre le concept "Ifc door type". Les assertions ① et ② de la règle (a) délivrent la même information. Cependant, elles explicitent le concept, "Ifc rel defines by type", servant d'ancrage aux propriétés "is typed by" et "relating type", ce qui rend la règle (a) plus longue.

— De même, les assertions ③ et ④ de la règle RAINS (a) sont équivalentes à l'assertion ❷ de la règle RAINS (b).

En proposant ces deux formulations RAINS équivalentes pour notre exigence, nous voulions mettre en évidence la possibilité qu'offre RAINS de chainer les propriétés. Au

niveau de la syntaxe formelle de RAINS (*listing* 4.1, page 80), cette concaténation de propriétés est représentée par le symbole `ChainOfObjectProperties`. Notons que seules des *object properties* (et donc pas de `datatype properties`) peuvent faire partie de la chaîne de propriétés. Ceci est dû au fait que partant d'une *datatype property*, on arrive nécessairement sur un littéral. Or, d'un littéral, qui est une simple valeur, on ne peut pas trouver, dans un graphe RDF, une propriété nous permettant d'atteindre une autre entité.
Nous concluons cette série d'exemples détaillés de règles RAINS, sur les deux remarques suivantes.

1. Nous avons reformulé l'exigence originale, lorsque cela était nécessaire, pour réorganiser la règle selon la structure du *logical pattern* c'est-à-dire *If . . . then*. Cependant, nous avons toujours écrit le conséquent sous une forme permettant d'exhiber l'objet de non-conformité. Par exemple "*If a door has multiple panels and none of its panels' width exceed 0.80 meter,* **then this door is not compliant**". Pour simplement respecter la contrainte imposée par RAINS de rendre explicite le *logical pattern*, cette règle aurait pu être reformulée de la façon suivante : "*If a door has multiple panels, then at-least one of the panels must have a minimal width of 0.80 meter.*". Cette formulation conserve dans la partie conclusion de la règle un marqueur d'obligation (*must have*). Pour la semi-formaliser (de manière litérale), il est nécessaire de disposer d'opérateurs de logique modale (obligation, interdiction, permission). Le langage RAINS ne faisant pas appel à la logique modale, contraint donc à reformuler les normes de manière à faire ressortir, non seulement le *logical pattern*, mais aussi à mettre en avant la détection de défaut :
   — Une obligation non respectée est un signe de défaut,
   — Une occurrence d'interdiction est un point de non-conformité,
   — Le respect ou non d'une permission ne permet pas de signaler la présence d'un défaut. C'est d'ailleurs la raison pour laquelle seules les prescriptions strictes (obligations et interdictions) sont concernées par notre travail (voir section 3.1.5).
2. En regardant la syntaxe formelle de RAINS, nous notons qu'il n'apparait pas de termes tels que ***verbe***, ***non commun***, ***déterminant*** etc. qui permettent à certains langages contrôlés de s'appuyer explicitement sur le vocabulaire d'une langue déterminée (Anglais, Français, etc.). RAINS ne manipule que des entités, des fragments de textes en langue naturel et quelques mot-clés exprimés de manière native en anglais. Cela permet à une phrase RAINS de pouvoir être écrite en plusieurs langues, tout en conservant la même sémantique. Pour illustrer ce propos, considérons le fragment d'ontologie du *listing* 4.2. Cette ontologie reprend des termes de IfcOWL (figure 4.1) et met en évidence le fait que chaque entité dispose de libellés (`rdfs:label`) en français (`@fr`) et en anglais (`@en`).

Listing 4.2 – Extrait de l'ontologie IfcOWL présentée en figure 4.1)

```
@prefix : <http://example.com/LiftingPlatformOntology#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.com/LiftingPlatformOntology#> rdf:type owl:Ontology .

:LiftingPlatform
        rdf:type owl:Class;
        rdfs:label "lifting platform"@en,
        "lift platform"@en,
        "plateforme élévatrice"@fr .

:NonCompliant
        rdf:type owl:Class;
        rdfs:label "non-compliant"@en,
        "non-conforme"@fr .

:ratedSpeed
        rdf:type owl:DatatypeProperty;
        rdfs:range xsd:double;
        rdfs:domain :LiftingPlatform;
        rdfs:label "rated speed"@en,
        "vitesse nominale"@fr .
```

Ainsi la règle RAINS de l'exemple 1 à savoir

`If the "rated speed" of a "Lifting platform" is "not" "less than or equal to" "0.15" m/s`

`Then it is "Non-compliant"2`

Pourrait aussi s'écrire, parce que `:LiftingPlatform` possède deux libellés de langue anglaise `"lifting platform"` et `"lift platform"` :

`If the "rated speed" of a "Lift platform" is "not" "less than or equal to" "0.15" m/s`

`Then it is "Non-compliant"2`

La même exigence pourrait être reprise ***en français*** ; dans ce cas, on obtiendrait

`Si la "vitesse nominale" de la "Platforme élévatrice" "n'est pas" "inférieure ou égale à" "0.15" m/s`

`Alors elle est "Non-conforme"2`

Sous réserve du fait que les entités de l'ontologie de domaine disposent de libellés en français, on se rend compte que RAINS est un langage contrôlé qu'on peut adapter en français en traduisant uniquement les mots clés du langage RAINS. Par mots clés, nous entendons les mots permettant d'introduire les assertions (*If* $\rightarrow$ ***Si***,

When → ***Lorsque***, *Then* → ***Alors***, etc.), les opérateurs de comparaison (*equal* → ***égal***, *greater than* → ***plus grand que***, etc.), les marques de négation (*is not* → ***n'est pas***, etc.), les marques de restriction universelle (*only* → ***seulement***, *exclusively* → ***exclusivement***, etc.). Vu le nombre réduit de ces termes (moins de cinquante), nous considérons que l'adaptation de RAINS, de sa langue native, l'anglais, vers une autre langue dont la syntaxe est similaire à celle de l'anglais (par exemple le français), est une opération simple à effectuer. De plus, peu importe la langue, les exigences RAINS conservent la même sémantique, la même expression formelle.

4.5 Sémantique formelle de RAINS

Au terme de la section 3.2.2 nous avons fait le choix du langage SPARQL pour représenter les normes. La sémantique de RAINS est directement alignée sur celle de SPARQL. Rappelons que SPARQL est un langage de requêtes et que chaque requête SPARQL se construit à l'aide de *graph patterns* ; un *graph pattern* étant formé d'un ou plusieurs *triple patterns*. Lorsque nous disons que la sémantique de RAINS est alignée sur celle de SPARQL, cela veut dire que chaque règle RAINS donne lieu à une requête SPARQL.
Une phrase RAINS se compose de zéro ou plusieurs annotations, d'un antécédent et d'un conséquent. Pour chacun de ces trois éléments, nous décrivons la manière dont ils sont interprétés.
Comme nous le verrons, toutes les annotations et assertions correctes d'un point de vue syntaxique, ne le sont pas nécessairement au regard de l'ontologie de domaine. Pour présenter ces cas où les annotations et assertions sont valides à la fois syntaxiquement et sémantiquement, nous introduisons les définitions ci-après.

Définition 1. Le ***range*** d'une propriété `p`, noté `range(p)`, est l'ensemble des classes `C` tel que, `C` est un descendant du `rdfs:range` de `p`. On a :
`range(p)` $= \{$`C`$|$ `C rdfs:subClassOf* Rp` $\wedge$ `p rdfs:range Rp`$\}$
Exemple : `range(:relatingType)` $= \{$`:IfcDoorType`$\}$, `range(:ratedSpeed)` $= \{$`xsd:double`$\}$

Définition 2. Le ***domain*** d'une propriété `p`, noté `domain(p)`, est l'ensemble des classes `C` tel que, `C` est un descendant du `rdfs:domain` de `p`. On a :
`domain(p)` $= \{$`C`$|$ `C rdfs:subClassOf* Dp` $\wedge$ `p rdfs:domain Dp`$\}$
Exemple : `domain(:relatingType)` $= \{$`:IfcRelsDefineByType`$\}$,
`domain(:ratedSpeed)` $= \{$`:LiftingPlatform`$\}$

Définition 3. **`Out-properties(C)`**, où `C` est une classe (exception faite des classes regroupant des littéraux), désigne l'ensemble des propriétés `p` telles qu'une instance de `C` peut être le sujet d'un triplet RDF ayant pour prédicat `p`. On a :

`out-properties(C)` $= \{$`p`$|$ `p rdfs:domain Dp` $\wedge$ `C rdfs:subClassOf* Dp`$\} = \{$`p`$|$ `C` $\in$ `domain(p)`$\}$
Exemple : `out-properties(:LiftingPlatform)` $= \{$`:ratedSpeed`, `:emergyControlDevice`, `:length`$\}$

Définition 4. **`In-properties(C)`**, où `C` est une classe, désigne l'ensemble des propriétés `p` telles qu'une instance de `C` peut être l'objet d'un triplet RDF ayant pour prédicat `p`. On a :
`in-properties(C)` $= \{$`p`$|$ `p rdfs:range Rp` $\wedge$ `C rdfs:subClassOf* Rp`$\} = \{$`p`$|$ `C` $\in$ `range(p)`$\}$
Exemple : `in-properties(xsd:double)` $= \{$`:length` , `:ratedSpeed`, `:panelWidth`$\}$

Définition 5. **`Out-properties(i)`**, où `i` est un individu, désigne l'ensemble des propriétés `p` telles que `i` peut être le sujet d'un triplet RDF ayant pour prédicat `p`. On a :
`out-properties(i)` $= \{$`p`$|$ `p` $\in$ `in-properties(C)` $\wedge$ `i rdfs:type C`$\}$

Définition 6. **`In-properties(i)`**, où `i` est un individu, désigne l'ensemble des propriétés `p` telles que `i` peut être l'objet d'un triplet RDF ayant pour prédicat `p`. On a :
`in-properties(i)` $= \{$`p`$|$ `p` $\in$ `in-properties(C)` $\wedge$ `i a C`$\}$
Exemple : `in-properties(:PUBLIC_BUILDING)` $= \{$`:buildingUse`$\}$

Définition 7. **`Out-properties(p)`**, où `p` est une propriété, désigne l'ensemble des propriétés `q` telles que, l'objet d'un triplet RDF ayant pour prédicat `p` peut être le sujet d'un triplet RDF ayant pour prédicat `q`. On a :
`Out-properties(p)` $= \{$`q`$|$ $\exists$`Cp` $\in$ `range(p)` $\wedge$ $\exists$`Cq` $\in$ `domain(q)` $\wedge$ `Cq rdfs:subClassOf* Cp`$\}$
Exemple : `out-properties(:relatingType)` $= \{$`:hasPropertySets`, `:operationType`$\}$

Définition 8. **`In-properties(p)`**, où `p` est une propriété, désigne l'ensemble des propriétés `q` telles que, l'objet d'un triplet RDF ayant pour prédicat `q` peut être le sujet d'un triplet RDF ayant pour prédicat `p`. On a :
`in-properties(p)` $= \{$`q`$|$ $\exists$`Cq` $\in$ `range(q)` $\wedge$ $\exists$`Cp` $\in$ `domain(p)` $\wedge$ `Cp rdfs:subClassOf* Cq`$\}$
$= \{$`q`$|$ `p` $\in$ `out-properties(q)`$\}$
Exemple : `in-properties(:operationType)` $= \{$`:relatingType`$\}$,
`in-properties(:panelWidth)` $= \{$`:hasPropertySets`$\}$

Définition 9. On dit de deux *datatype properties* `p` et `q` qu'elles sont comparables, et on note **`comparable(p,q)`**, si l'intersection de leur *range* est non-vide ; autrement dit, elles prennent leurs valeurs dans des classes de littéraux semblables. De manière formelle, on a :
`comparable(p,q)` $\implies$ $\exists$`L`$|$ `range(p)` $\cap$ `range(q)` $\neq \emptyset$
Par exemple, on a `comparable(:panelWidth, :length)` car les propriétés `:panelWidth` et `:length` ont toutes les deux pour *range* le singleton {`xsd:double`}.

Parce que nous les avons trouvé significatifs, ces définitions reprennent les dénominations utilisées dans les travaux de Mazzeo & Zaniolo [2016]. Cependant, la définition de chaque terme est propre au travail que nous présentons ici et peut présenter des différences par rapport à celles fournies par Mazzeo & Zaniolo.

4.5.1 Les Annotations

Dans ses travaux sur la modélisation du contrôle de conformité, Yurchyshyna [2009] propose de représenter les règles comme des instances de la classe `:Rule` (voir figure 4.2).

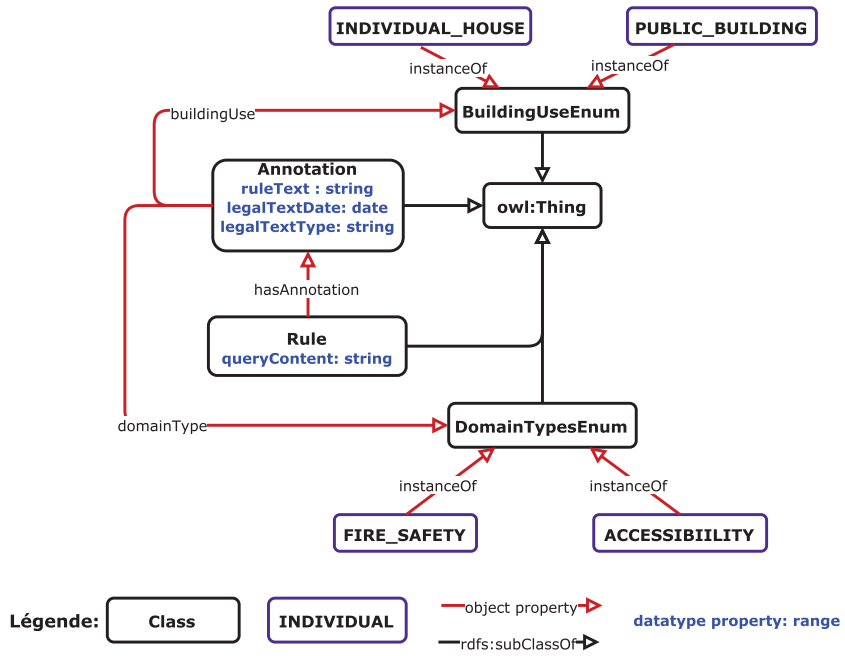


FIGURE 4.2 – Extrait de l'ontologie d'annotations sémantiques de règles (annoComplexe.owl) [Yurchyshyna 2009, Annexe 3]

Ainsi, la règle écrite sous forme de requête SPARQL sera un attribut, via la *datatype property* `:queryContent`, de cette instance de `:Rule`. Par ailleurs à chaque instance de la classe `:Rule` sera rattachée une instance de la classe `:Annotation`. Comme nous l'avons déjà mentionné, une annotation permet de définir le contexte de la règle. Par exemple, le ***type du texte réglémentaire*** - propriété `:legalTextType` - dans lequel la règle a été extraite (voir figure 2.4), la ***date de publication*** de ce document - propriété `:legalTextDate`, le ***texte en langue naturel*** de la règle - propriété `:ruleText`, et plus spécifiquement au domaine de la Construction, des informations comme l' ***usage du bâtiment*** (`:buildingUse`) ou le ***domaine*** (`:domainType`) concerné par la règle, entre autres.
Aussi, une annotation RAINS donne lieu à une instance, par exemple `:ANNOTATION_X`, de la classe `:Annotation`. Pour cette instance `:ANNOTATION_X` et pour ***chaque couple***

(AnnotationName, AnnotationValue) - par exemple (:annotationNameY, :ANNOTATION_VALUE_Z), on aura un triplet RDF. Ce qui donnera

```
:ANNOTAION_X rdf:type :Annotation;
        :annotationNameY :ANNOTAION_VALUE_Z.
```

Appliqué à l'annotation de l'exemple 2 de la section 4.4, que nous reprenons ci-dessous

```
@ Building use(Public building)
```

on obtient le graphe RDF suivant :

```
:AnnotationRegle2 rdf:type :Annotation;
                :buildingUse :PUBLIC_BUILDING.
```

Il est important de noter que tout couple (:AnnotationName, :AnnotationValue) n'est pas nécessairement valide. Pour qu'un tel couple soit valide, il est ***nécessaire et suffisant que*** : AnnotationName soit le label d'une *datatype* ou d'une *object property*, et que :

1. :Annotation $\in$ domain(:AnnotationName)
2. :AnnotationName $\in$ in-properties(C), avec $\{$:AnnotationValue a C$\}$

4.5.2 L'Antécédent et le Conséquent

L'antécédent et le conséquent font partie de la requête SPARQL (qui n'est rien d'autre que l'exigence formelle) proprement dite. L'antécédent donnera lieu à une ***clause WHERE*** tandis que le conséquent donnera lieu à une ***clause CONSTRUCT***. À l'intérieur de chacune de ces clauses, on trouvera des *graph patterns* qui dépendent essentiellement du type d'assertions RAINS que l'on retrouve dans l'antécédent et dans le conséquent. Pour chaque type d'assertion, nous fournissons le *graph pattern* correspondant. Pour cela, nous allons rappeler l'expression formelle de chaque type d'assertion telle qu'exposée dans le *listing* 4.1, sans mention des symboles Text - ces derniers servent à insérer du langage naturel dans des règles RAINS mais n'ont aucune sémantique formelle. Ensuite, pour chacune des déclinaisons possibles (c'est-à-dire en fonction de la présence ou non des éléments optionnels) de cette expression, nous donnerons le *graph pattern* correspondant. Une assertion RAINS utilise des termes du domaine. Comme nous l'avons déjà souligné, ces derniers sont encapsulés par les symboles Concept, BooleanDatatypeProperty, DatatypeProperty, ObjectProperty et Individual dans la syntaxe EBNF de RAINS. Dans une requête SPARQL valide, ces entités doivent appraître sous forme d'URI. Nous représenterons cette URI sous la forme **:Entite** (par exemple :Concept, :Individual, etc.). Lorsqu'une assertion utilisera un type d'entité plus d'une fois, alors nous ajouterons un entier à cette URI fictive. Par exemple :Concept1, :Concept2, dans une assertion utilisant deux fois une entité de type Concept ; :Concept1 étant le premier concept apparaissant dans l'assertion et :Concept2 le second (de la gauche vers la

droite). Pour un littéral, `:Literal` désigne sa classe et `Litteral` le littéral proprement dit (c'est-à-dire sa valeur). Par ailleurs, nous présentons, pour chaque type d'assertion ***syntaxiquement correcte***, ses cas de ***validité***, autrement dit les cas où cette assertion est en accord avec la sémantique du graphe que décrit l'ontologie du domaine. Rappelons que la syntaxe de RAINS concerne non seulement l'agencement des entités les unes par rapport aux autres, mais aussi le fait que certaines entités se réfèrent à des éléments existants dans l'ontologie de domaine (`Concept`, `Individual`, `DatatypeProperty`, `ObjectProperty` et `BooleanDatatypeProperty`), ou font partie d'une liste de termes (opérateurs de comparaisons, marques de négations et de quantification universelles) ou alors sont des littéraux (`Literal`).

Voici donc pour chaque type d'assertion, sa définition, le *graph pattern* SPARQL correspondant ainsi que les critères de validité.

Single Concept Assertion

— **Définition :**
`SingleConceptAssertion = Concept`

— ***Graph Pattern* :**

```
{?x rdf:type :Concept}
```

— **Validité.** Toute *single concept assertion* ayant une syntaxe correcte, c'est-à-dire le `Concept` existe dans l'ontologie de domaine, est toujours valide.

Double Concept Assertion

— **Définition :**
`DoubleConceptAssertion = Concept, Neg?, SimpleConcept`

— **Déclinaison 1 :**
`DoubleConceptAssertion = Concept, SimpleConcept`
***Graph Pattern* :**

```
{?x rdf:type :Concept.
 ?x rdf:type :SimpleConcept}
```

— **Déclinaison 2 :**
`DoubleConceptAssertion = Concept, Neg, SimpleConcept`
***Graph Pattern* :**

```
{?x rdf:type :Concept.
 MINUS{?x rdf:type :SimpleConcept} }
```

— **Validité.** On peut remarquer que quelle que soit la déclinaison, la variable utilisée pour se référer au premier concept (`?x rdf:type :Concept`) est la même que celle utilisée pour se référer au second concept (`?x rdf:type :SimpleConcept`). Ce type d'assertion est toujours valide.

Data Assertion

— **Définition :**

`DataAssertion = Concept, Neg?, ChainOfObjectProperties?, DatatypeProperty, Neg?, Comparator, Quantity`

Sachant que le symbole `Quantity` peut être soit un littéral soit un attribut de concept, ce cas se décline en 16 sous-cas. Pour réduire le nombre de cas, nous supposerons que la Chaine de propriétés `ChainOfObjectProperties` est toujours présente et est remplacée par une occurrence de `ObjectProperty` (voir ligne ❶ du *listing* 4.1 à la page 80).

— **Déclinaison 1 :**

`DataAssertion = Concept, ObjectProperty, DatatypeProperty, Comparator, Literal`

GraphPattern

```
{?x rdf:type :Concept.
 ?x :ObjectProperty ?y.
 ?y :DatatypeProperty ?z.
 FILTER(?z comparator Literal)}
```

— **Déclinaison 2 :**

`DataAssertion = Concept, ObjectProperty, DatatypeProperty, Neg, Comparator, Literal`

***Graph Pattern* :**

```
{?x rdf:type :Concept.
 ?x :ObjectProperty ?y.
 MINUS{ ?y :DatatypeProperty ?z.
        FILTER(?z comparator Literal)} }
```

— **Déclinaison 3 :**

`DataAssertion = Concept, Neg, ObjectProperty, DatatypeProperty, Comparator, Literal`

***Graph Pattern* :**

```
{?x rdf:type :Concept.
 MINUS{ ?x :ObjectProperty ?y.
```

```
        ?y :DatatypeProperty ?z.
        FILTER(?z comparator Literal)} }
```

Validité (déclinaisons 1 à 3). Pour ces trois premières déclinaisons, les conditions de validité sont les mêmes ; la négation n'est qu'un opérateur et donc n'a pas d'influence sur la validité (rappelons que la validité représente le respect de l'assertion vis-à-vis du schéma de l'ontologie du domaine) :

1. :Concept ∈ domain(:ObjectProperty)
2. :ObjectProperty ∈ in-properties(:DatatypeProperty)
3. :DatatypeProperty ∈ in-properties (:Literal)

— **Déclinaison 4 :**

DataAssertion = Concept, ObjectProperty, DatatypeProperty, Comparator, DatatypeProperty, Concept

Graph Pattern

```
{?x rdf:type :Concept1.
 ?x :ObjectProperty ?y.
 ?y :DatatypeProperty1 ?val1.
 ?z rdf:type :Concept2.
 ?z :DatatypeProperty2 ?val2.
 FILTER(?val1 comparator ?val2)}
```

— **Déclinaison 5 :**

DataAssertion = Concept, ObjectProperty, DatatypeProperty, Neg, Comparator, DatatypeProperty, Concept

Graph Pattern

```
{?x rdf:type :Concept1.
 ?x :ObjectProperty ?y.
 MINUS{ ?y :DatatypeProperty1 ?val1.
        ?z rdf:type :Concept2.
        ?z :DatatypeProperty2 ?val2.
        FILTER(?val1 comparator ?val2)} }
```

— **Déclinaison 6 :**

DataAssertion = Concept, Neg, ObjectProperty, DatatypeProperty, Comparator, DatatypeProperty, Concept

Graph Pattern

```
{?x rdf:type :Concept1.
  MINUS{ ?x :ObjectProperty ?y.
         ?y :DatatypeProperty1 ?val1.
```

```
?z rdf:type :Concept2.
?z :DatatypeProperty2 ?val2.
FILTER(?val1 comparator ?val2)} }
```

Validité (déclinaisons 4 à 6).

1. :Concept1 ∈ domain(:ObjectProperty)
2. :ObjectProperty ∈ in-properties(:DatatypeProperty1)
3. :Concept2 ∈ domain(:DatatypeProperty2)
4. comparable(:DatatypeProperty1, :DatatypeProperty2)

Inverted Data Assertion

Lors de la présentation des différentes assertions RAINS à la section 4.2, nous avons souligné les similitudes syntaxiques entre les *data assertions* et les *inverted data assertions*. La principale différence entre les deux est que dans le second cas, la propriété est placée avant le concept (exemple : "Lifting plaform" has a "rated speed" *vs* "rated speed" of a "Lifting platform"). Cette différence de syntaxe n'entraîne pas de différence sémantique. Une *inverted data assertion* a donc le même *graph pattern* qu'une *data assertion* de même "déclinaison" et ses conditions de validité sont les mêmes.

Boolean Assertion

— **Définition :**
BooleanAssertion = Concept, Neg?, ChainOfObjectProperties?, BooleanDatatypeProperty
La présence ou non d'une chaîne d'*object properties* a le même impact que dans le cas d'une *data assertion* (voir sections précédentes). Nous présentons uniquement les déclinaisons qui nous font voir les spécificités dues à la *boolean datatype property*.

— **Déclinaison 1 :**
BooleanAssertion = Concept, BooleanDatatypeProperty
***Graph Pattern* :**

```
{?x rdf:type :Concept.
 ?x :BooleanDatatypeProperty "true" }
```

— **Déclinaison 2 :**
BooleanAssertion = Concept, Neg, BooleanDatatypeProperty
***Graph Pattern* :**

```
{?x rdf:type :Concept.
```

```
MINUS{?x :BooleanDatatypeProperty "true"} }
```

Validité. La négation n'a pas d'influence sur la validité d'une *boolean assertion*

1. :Concept ∈ domain(:BooleanDatatypeProperty)

Object Assertion

— **Définition**

ObjectAssertion = OrderedObjectAssertion | InvertedObjectAssertion | OrderedObjectAssertionWithIndiv | InvertedObjectAssertionWithIndiv;

Une *object assertion* peut être de l'un des quatre sous-types mentionnés dans la définition ci-dessus. Pour les expressions utilisant une ChainOfObjectproperties, nous supposerons cette dernière composée de deux *object properties*. Nous présentons les *graph patterns* en fonction de chaque sous-type.

Sous-cas de OrderedObjectAssertion. Nous regroupons les déclinaisons ayant les mêmes critères de validité.

— **Déclinaison 1.1 :**

ObjectAssertion = Concept, ObjectProperty, ObjectProperty, Concept

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 ?x :ObjectProperty1 ?y.
 ?y :ObjectProperty2 ?z.
 ?z rdf:type :Concept2}
```

— **Déclinaison 1.2 :**

ObjectAssertion = Concept, Neg, ObjectProperty, ObjectProperty, Concept

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 MINUS{ ?x :ObjectProperty1 ?y.
        ?y :ObjectProperty2 ?z.
        ?z rdf:type :Concept2.} }
```

— **Déclinaison 1.3 :**

ObjectAssertion = Concept, ObjectProperty, ObjectProperty, UniversalRestrictionKeyword, Concept

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 ?x :ObjectProperty1 ?y.
 ?y :ObjectProperty2 ?z.
 ?z rdf:type :Concept2.
```

```
?subConcept2 rdfs:subClassOf* :Concept2.
FILTER NOT EXISTS {?x :ObjectProperty2 ?z2. ?z2 a ?t.
       FILTER((?z2 != ?z) &&(?t NOT IN (?subConcept2)))}}
```

— **Déclinaison 1.4 :**

ObjectAssertion = Concept, Neg, ObjectProperty, ObjectProperty, UniversalRestrictionKeyword, Concept

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 MINUS{ ?x :ObjectProperty1 ?y.
        ?y :ObjectProperty2 ?z.
        ?z rdf:type :Concept2.
        ?subConcept2 rdfs:subClassOf* :Concept2.
        FILTER NOT EXISTS {?x :ObjectProperty2 ?z2. ?z2 a ?t.
         FILTER((?z2 != ?z)&&(?t NOT IN (?subConcept2)))}} }
```

— **Déclinaison 1.5 :**

ObjectAssertion = Concept, ObjectProperty, ObjectProperty, Comparator, Integer, Concept

***Graph Pattern* :**

```
{ SELECT ?x (COUNT (distinct ?c2) as ?numberOfC2){
        ?x rdf:type :Concept1.
        ?x :ObjectProperty1 ?y.
        ?y :ObjectProperty2 ?c2.
        ?c2 rdf:type :Concept2.
        } GROUP BY ?x
          HAVING (?numberOfC2 comparator Integer) }
```

— **Déclinaison 1.6 :**

ObjectAssertion = Concept, Neg, ObjectProperty, ObjectProperty, Comparator, Integer, Concept

***Graph Pattern* :**

```
{ ?x rdf:type :Concept1.
  MINUS{
        SELECT ?x (COUNT (distinct ?c2) as ?numberOfC2){
        ?x :ObjectProperty1 ?y.
        ?y :ObjectProperty2 ?c2.
        ?c2 rdf:type :Concept2.
        } GROUP BY ?x
          HAVING (?numberOfC2 comparator Integer) } }
```

Validité (déclinaisons 1.1 à 1.6). Une fois de plus, nous mentionnons que les validités de ces déclinaisons ne dépendent que des relations entre les entités présentes dans l'ontologie de domaine et non des éléments syntaxiques tels que les marques de négations ou de restrictions.

1. :Concept1 ∈ domain(:ObjectProperty1)
2. :ObjectProperty1 ∈ in-properties(:ObjectProperty2)
3. :Concept2 ∈ range(:ObjectProperty2)

— **Déclinaison 1.7 :**

ObjectAssertion = Concept, ObjectProperty, ObjectProperty, Concept, DatatypeProperty, Comparator, Literal;

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 ?x :ObjectProperty1 ?y.
 ?y :ObjectProperty2 ?z.
 ?z rdf:type :Concept2.
 ?z :DatatypeProperty ?v. FILTER(?v comparator Litteral)}
```

— **Déclinaison 1.8 :**

ObjectAssertion = Concept, Neg, ObjectProperty, ObjectProperty, Concept, DatatypeProperty, Comparator, Literal;

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 MINUS{
 ?x :ObjectProperty1 ?y.
 ?y :ObjectProperty2 ?z.
 ?z rdf:type :Concept2.
 ?z :DatatypeProperty ?v. FILTER(?v comparator Literal)} }
```

Validité (déclinaisons 1.7 et 1.8).

1. :Concept1 ∈ domain(:ObjectProperty1)
2. :ObjectProperty1 ∈ in-properties(ObjectProperty2)
3. :Concept2 ∈ range(:ObjectProperty2)
4. :Concept2 ∈ domain(:DatatypeProperty)
5. :DatatypeProperty ∈ in-properties (:Literal)

D'autres déclinaisons existent pour OrderedObjectAssertion. Leur *graph pattern* et leur conditions de validité sont similaires à ce qui a été présenté pour les déclinaisons 1.1 à 1.8.

Sous-cas de `InvertedObjectAssertion`.

— **Déclinaisons 2 :**

Du point de vue syntaxique, la principale différence entre ce type d'assertion et les `OrderedObjectAssertion` réside dans la position de l'*object property* par rapport au concept caractérisé. Cette nuance syntaxique n'entraine aucune différence sémantique.

Sous-cas de `OrderedObjectAssertionWithIndiv`.

— **Déclinaison 3.1 :**

`ObjectAssertion = Concept, ObjectProperty, ObjectProperty, Individual`

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 ?x :ObjectProperty1 ?y.
 ?y :ObjectProperty2 :Individual}
```

— **Déclinaison 3.2 :**

`ObjectAssertion = Concept, Neg, ObjectProperty, ObjectProperty, Individual`

***Graph Pattern* :**

```
{?x rdf:type :Concept1.
 MINUS{ ?x :ObjectProperty1 ?y.
        ?y :ObjectProperty2 :Individual} }
```

Validité (déclinaisons 3.1 et 3.2).

1. `:Concept1` ∈ `domain(:ObjectProperty1)`
2. `:ObjectProperty1` ∈ `in-properties(ObjectProperty2)`
3. `:ObjectProperty2` ∈ `in-properties (:Individual)`

Sous-cas de `InvertedObjectAssertionWithIndiv`.

— **Déclinaisons 4 :**

Dans ce cas aussi, la sémantique est similaire à celle de `OrderedObjectAssertionWithIndiv`.

4.5.3 Exemples de règles SPARQL

Nous reprenons les trois exemples de la section 4.4 en page 84 et nous donnons les représentations formelles correspondantes au regard de la sémantique de RAINS.

Exemple 1 : *"The rated speed of the lifting platform shall not be greater than 0,15 m/s"*

Pour rappel, la règle RAINS obtenue à partir de cette règle est :

```
If the "rated speed" of a "Lifting platform" is "not" "less than or equal to"
```

```
"0.15" m/s
Then it is "Non-compliant"2
```

Elle se convertie en la requête SPARQL :

```
CONSTRUCT{?x rdf:type :NonCompliant}
WHERE{?x rdf:type :LiftingPlatform.
        MINUS{ ?x :ratedSpeed ?y. FILTER(?y <= 0.15)} }
```

On remarque que la co-référence de l'entité `"Non-Compliant"2` à l'entité `"Lifting Platform"` permet de déduire que la variable `?x` associée au concept `:LiftingPlatform` doit aussi être associée à `:NonCompliant`.
Il est aussi important de noter qu'il y a une différence, d'un point de vue formelle, entre les deux assertions ci-dessous :

1. `If the "rated speed" of a "Lifting platform" is "not" "less than or equal to" "0.15" m/s`
2. `If the "rated speed" of a "Lifting platform" is "greater than" "0.15" m/s`

En effet, la transformation d'assertions RAINS en SPARQL *graph pattern* se fonde sur l'***hypothèse du monde clos***. Ainsi, pour se conformer à la seconde assertion RAINS, il faut prouver que la vitesse nominale de la plateforme est supérieure à 0.15 ; ce qui suppose que cette dernière est fournie. Tandis que pour la première assertion, il faut déboucher sur un échec en essayant de démontrer que la vitesse est inférieure ou égale à 0.15, ce qui peut se faire de deux manières : (*i*) Soit obtenir la vitesse de la plateforme et constater qu'elle n'est pas inférieure ou égale à la valeur 0.15, (*ii*) Soit ne pas avoir à disposition la valeur de la vitesse nominale et donc se trouver dans l'incapacité d'affirmer que cette vitesse est inférieure ou égale à 0.15, et ainsi conclure qu'elle n'est pas inférieure ou égale à la valeur seuil.
Comme nous l'avons souligné au chapitre 3, nous avons choisi d'utiliser la négation par l'échec parce que ce travail est effectué dans le cadre de la conformité réglementaire qui elle-même est étroitement liée avec les aspects de sécurité. Ainsi, il faudrait que, lorsqu'on affirme qu'un produit est conforme, c'est parce que l'on est parvenu à trouver des faits explicites qui attestent de sa conformité. Par conséquent, l'absence d'informations sur un aspect du produit concourt à sa non-conformité.
Ainsi, chaque fois qu'un expert métier écrira une règle RAINS, il faudra qu'il ait à l'idée l'importance de l'utilisation (ou de la non-utilisation) des marques de négation. Cela n'exige pas de ce dernier d'avoir une maîtrise du concept de négation par l'échec, mais de savoir par exemple si un attribut du schéma du domaine, est obligatoirement fourni ou optionnel.

Exemple 2 : *"In buildings with public access, the platform length shall not be less than 1 400 mm, to enable sufficient space for an attendant."*

Cette exigence a donné lieu à la règle RAINS ci-après :

```
@ Building Use(Public building)
Given a "Lifting platform" with a "length" that is "not" "greater than or equal to" "1400" mm
Then it is "Non-compliant"1
```

Comme nous l'avons mentionné en section 4.5.1, les annotations donnent lieu à une instance de la classe `:Annotation` qui sera associée aux valeurs de l'annotation. Ce qui donne le résultat suivant dans cet exemple :

```
:AnnotationRegle2 rdf:type :Annotation;
                  :buildingUse :PUBLIC_BUILDING.
```

Le corps de la règle (antécédent et conséquent) RAINS sera transformé en la requête SPARQL suivante :

```
CONSTRUCT{?x rdf:type :NonCompliant}
WHERE{?x rdf:type :LiftingPlatform.
        MINUS{?x :length ?y. FILTER(?y >= 1400)} }
```

Exemple 3 : *"For doors with multiple panels, one of the panels must have a minimal width of 0.80 meter."*

Nous avons transformé cette exigence en la règle RAINS ci-dessous.

```
If an "Ifc door" "is typed by"."relating type" an "Ifc door type"
And the "operation type" of that "Ifc door type"4 is "DOUBLE_DOOR_FOLDING"
And that "Ifc door type"4 "has property sets"."panel width" that is not "at least" "800" mm
Then this "Ifc Door"1 is "Non-compliant"
```

Lorsque cette dernière est transformée en requête SPARQL, on obtient le résultat suivant :

```
CONSTRUCT{?x rdf:type :IfcDoor. ?x rdf:type :NonCompliant}
WHERE{
 ?x rdf:type :IfcDoor.
 ?x :isTypedBy ?y. ?y :relatingType ?z. ?z rdf:type :IfcDoorType.
 ?z :operationType :DOUBLE_DOOR_FOLDING.
 ?z :hasPropertySets ?t.
 MINUS{ ?t :panelWidth ?u. FILTER(?u >= 800)}}
```

4.6 Typologie et position de RAINS sur l'échelle PENS

Au chapitre 3, nous avons présenté les échelles proposées par Kuhn [2014] pour classifier les langages contrôlés. Dans cette section, nous utilisons ces mêmes échelles pour caractériser RAINS.

4.6.1 Typologie de RAINS

Kuhn a proposé neuf types de langages qui sont résumés dans le tableau 3.2. En se basant sur ce tableau, nous pouvons dire que RAINS est un langage de type :

— **F**, car il permet de faciliter une représentation formelle et exécutable du langage naturel ;
— **W**, parce que destiné à l'écriture ;
— **D**, car conçu en étudiant un domaine bien précis. Notons toutefois que RAINS est un langage dédié à l'écriture semi-formelle de règles pour le contrôle de conformité. Le contrôle de conformité n'étant pas spécifique au secteur de la Construction, RAINS peut être réutilisé dans d'autres domaines.
— **A** parce qu'il découle de travaux académiques.

4.6.2 RAINS sur l'échelle PENS

Après avoir typé RAINS, nous nous intéressons à son positionnement sur les axes de l'espace ⟨Précision, Expressivité, Naturalité et Simplicité⟩ tels que proposés dans [Kuhn 2014]. Nous avons présenté l'espace PENS à la section 3.4.2 (pages 58 et suivantes).

1. **Précision.** ***RAINS est un langage de niveau*** P^5. Autrement dit, c'est un langage à sémantique fixe. Comme nous l'avons vu, RAINS dispose d'une syntaxe et d'une sémantique formelle. Chaque phrase RAINS est formalisable de manière automatique sans faire appel à des heuristiques ou à des connaissances externes.
2. **Expressivité.** ***RAINS est un langage d'expressivité*** E^3. En effet, si l'on considère tous les types d'assertions RAINS dans toutes leur déclinaisons, on se rend compte qu'à l'aide du langage RAINS, on peut formuler des énoncés exprimant :
   (a) La quantification universelle sur les individus,
   (b) Les relations (binaires) entre plusieurs entités,
   (c) Les énoncés sous forme de règle (c'est-à-dire sous la forme ***si*** … ***alors***),
   (d) La négation.
3. **Naturalité.** Une phrase RAINS se compose d'annotations et d'une règle dont l'antécédent et le conséquent sont explicites. Cette structure fait perdre à RAINS des

points sur chemin de la naturalité. Cependant, l'antécédent et le conséquent d'une phrase RAINS se composent d'assertions dont la structure est directement calquée sur l'organisation des termes au sein des phrases en langage naturel. Par ailleurs, ces assertions sont construites principalement avec des termes du domaine concerné par la règle. La naturalité d'une assertion RAINS prend ainsi avantage de celle des libellés de ces termes. Toutefois, nous avons vu que ces termes sont signalés dans une assertion RAINS à l'aide de guillemets. Aussi les co-références sont explicites et signalées par des nombres permettant de signaler à quelle entité on fait référence. Tous ces éléments font perdre à RAINS des parts de naturalité. Néanmoins, RAINS permet de lier les entités par des expressions en langage naturel, sans restriction. Cela permet, entre autre, d'atténuer le caractère non-naturel de certains libellés que l'on peut retrouver dans des ontologies de domaine. Pour toutes ces raisons, nous considérons ***RAINS comme un langage de niveau*** N^3 (c'est-à-dire un point de moins que Gherkin qui est classé N^4). En d'autres termes, c'est un langage contrôlé avec une prédominance d'éléments naturels, mais n'utilisant pas de phrases naturelles.

4. **Simplicité.** La syntaxe du langage RAINS dont un extrait est présenté dans le *listing* 4.1 tient sur deux pages. Lorsqu'à cela nous ajoutons sa sémantique (voir section 4.5) et des éléments du langage SPARQL, nous obtenons une description sur plus de dix pages. Nous pouvons donc conclure avec Kuhn [2014] que ***RAINS est un langage de niveau de simplicité*** S^3.

4.7 Comparaisons des syntaxes de RAINS, PENG-D et Metalog PNL

À l'aide d'une série d'exemples de règles métiers, nous présentons les différences entre RAINS et les langages PENG-D et Metalog PNL. Ce sont deux langages contrôlés ortientés Web Sémantique et permettant d'écrire des règles métiers. De plus, tous les deux sont des langages à sémantique fixe (de niveau P^5 sur l'échelle PENS).

4.7.1 L'utilisation des entités

Règle métier en langage naturel
"The rated speed of the lifting platform shall not be greater than 0.15 m/s"

PENG-D

```
If the rated-speed of a lifting-platform is greater-than 0.15 then that
lifting-platform is non-compliant.
```

Metalog PNL

```
LIFTINGPLATFORM represents
"http://example.com/LiftOnto#LiftingPlatform".
RATEDSPEED represents "http://example.com/LiftOnto#ratedSpeed".
IS represents the verb "be" from the definitions in
"http://www.relationships.example.org/verbs".
NONCOMPLIANT represents "http://example.com/LiftOnto#NonCompliant".

If a LIFTINGPLATFORM has a RATEDSPEED that is greater than "0.15" then that
LIFTINGPLATFORM IS NONCOMPLIANT.
```

RAINS

```
If a "Lifting platform" "rated speed" "is greater than" "0.15" m/s
Then it is "Non-compliant"1
```

On peut remarquer que :

— Metalog PNL nécessite de redéfinir des alias pour les termes de l'ontologie de domaine. Pour une ontologie de plusieurs milliers de termes, ça fait autant d'alias à définir. Pour l'expert métier, cela constitue un travail supplémentaire et cela l'oblige à ce souvenir de ces alias en plus du schéma de l'ontologie de domaine. Par ailleurs, pour une personne venant lire des règles en PNL, elle est obligée de prendre connaissance de la définition des alias pour comprendre ces règles.

— Les trois langages font appel à des mots réservés. Dans le cas de RAINS, ces mots sont les délimiteurs de règles (`If`, `Then`, `And`, etc.), les marqueurs de négation et de quantification universelle ainsi que les prédicats de comparaison. Pour PENG-D, et Metalog PNL, en plus des types de mots réservés dans RAINS, il faut ajouter les déterminants (`the`, `a`, `an`), les auxiliaires (`have`, `be`), les prépositions (`of`), les pronoms (`that`).

— Les co-références dans les cas de PENG-D et PNL sont résolues en se basant sur l'article défini `the` ou sur le pronom `that`. Ces deux langages utilisent l'heuristique selon laquelle un terme précédé de `the` ou `that` fait référence à un terme ayant le même libellé et mentionné précédemment dans la règle. Ainsi, dans les exemples ci-dessus les analyseurs de PENG-D et PNL déduisent que la seconde mention de "lifting platform" fait référence à la première occurrence de "lifting platform". Dans le cas de RAINS, les co-références s'appuient sur les entiers placés à la suite des entités.

— Dans les phrases en PENG-D et en PNL, tous les mots sont porteurs de sens et sont décodés par les analyseurs respectifs de ces deux langages. Dans le cas de RAINS, seules les termes entre guillemets sont décodés. Dans les phrases en PNL, les guillemets permettent de signaler les littéraux.

4.7.2 Les informations descriptives

Règle métier en langage naturel
"In buildings with public access, the platform length shall not be less than 1400 mm, to enable sufficient space for an attendant."

PENG-D

```
If the length of a lifting-platform is less-than 1400 then that lifting-platform
is non-compliant.
```

Metalog PNL (On considère acquis les alias de l'exemple précédent)

```
LENGTH represents "http://example.com/LiftOnto#length".

If a LIFTINGPLATFORM has a LENGTH that is less than "1400" then that LIFTINGPLATFORM
IS NONCOMPLIANT.
```

RAINS

```
@ Building Use(Public building)
Given a "Lifting platform" with a "length" that is "not" "greater than or equal
to" "1400" mm
Then it is "Non-compliant"1
```

On peut remarquer que dans les langages PENG-D et PNL on a une perte d'informations. L'information dénotée par l'expression "In buldings with public access" est un élément contextuel. Elle ne rentre pas en compte dans le corps de la règle. PENG-D et PNL ne dispose pas de mécanisme pour représenter les informations contextuelles. RAINS propose des annotations pour accoler des informations descriptives aux règles.

4.7.3 La verbosité

Règle métier en langage naturel
"For doors with multiple panels, one of the panels must have a minimal width of 0.80 meter."

PENG-D

```
If an ifc-door is-typed-by an ifc-rel-defines-by-type that relating-type an
ifc-door-type that operation-type is double-door-swing and the ifc-door-type
has-property-sets an ifc-door-panel-properties that panel-width is less-than
800 then the ifc-door is non-conform
```

Metalog PNL (On considère acquis les alias des exemples précédents)

```
DOOR represents "http://example.com/LiftOnto#IfcDoor".
```

```
ISTYPEDBY represents "http://example.com/LiftOnto#isTypedBy".
DOORTYPE represents "http://example.com/LiftOnto#IfcDoorType".
DEFINEDTYPE represents "http://example.com/LiftOnto#IfcRelDefinesByType".
RELATINGTYPE represents "http://example.com/LiftOnto#relatingType".
OPERATION represents "http://example.com/LiftOnto#operationType".
DOUBLEDOOR represents "http://example.com/LiftOnto#DOUBLE_DOOR_SWING".
HASPROPERTIES represents "http://example.com/LiftOnto#hasPropertySets".
SETOFPROPERTIES represents "http://example.com/LiftOnto#IfcDoorPanelProperties".
PANELWIDTH represents "http://example.com/LiftOnto#panelWidth".

If a DOOR ISTYPEDBY a DEFINEDTYPE that RELATINGTYPE a DOORTYPE that OPERATION
IS DOUBLEDOOR and the DOORTYPE HASPROPERTIES a SETOFPROPERTIES that PANELWIDTH
is less than "800" then that DOOR IS NONCOMPLIANT.
```

RAINS

```
If an "Ifc door" "is typed by"."relating type" an "Ifc door type"
And the "operation type" of that "Ifc door type"4 is "DOUBLE_DOOR_FOLDING"
And this "Ifc door type"4 "has property sets"."panel width" that is "not" "at
least" "800" mm
Then this "Ifc Door"1 is "Non-compliant"
```

Dans cet exemple, on note que RAINS permet spécifier les conditions à vérifier dans l'antécédent en évitant une grande verbosité. La verbosité est atténuée par le mécanisme des chaînes de propriétés et l'obligation d'un recours à des assertions atomique. Dans PEND-D et Metalog PNL, on est obligé de concaténer ces conditions, peu importe leur nombre, sans interruptions. Aussi on est obligé d'expliciter toutes les entités (par exemple `Ifc-rel-defines-by-type` et `Ifc-door-panel-properties`) qui sont sous-entendues dans la règle en LN. Cela a pour conséquence une répétition de la conjonction de coordination `and` et du pronom `that` qui diminue la naturalité, voire la compréhension de la règle. Par ailleurs, dans un règle utilisant des entités ayant des libellés non-naturels (par exemple `Ifc-rel-defines-by-type`), la définition des alias dans Metalog PNL permet de renommer un libellé non-naturel par un nouveau nom plus naturel.

4.7.4 Utilisation libre du langage naturel

La possibilité d'utiliser librement des expressions en langage naturel, dans une règle RAINS, est un mécanisme important de RAINS. PENG-D et Metalog PNL n'offrent pas cette possibilité. Ces expressions naturelles :

— Agissent comme des commentaires pour améliorer la compréhension des règles. Pour illustration, reprenons les trois exemples que nous avons fournir dans cette section 4.7. Chacun d'eux mentionne une quantité (`0.15` dans le premier, `1400`

dans le deuxième et `800` dans le troisième). Dans les versions PENG-D et PNL de ces exemples, il n'est pas possible de renseigner l'unité de ces trois quantités. Cela est dû au fait que chaque terme présent dans une règle PENG-D ou PNL doit avoir une interprétation formelle. Or dans RAINS, seules les entités, explicitement signalés par des guillemets sont interprétés. Cela permet dans une règle RAINS, de préciser à côté d'une quantité l'unité de mesure correspondante. Cette précision n'aura de sens que pour les lecteurs humains de la règle.

— Agissent comme des liants pour améliorer la naturalité des règles de manière générale, et des règles utilisant des termes non-naturels, plus spécifiquement.

— Facilitent la traduction des règles RAINS. En effet, le recours à une utilisation optionnelle des expressions en langage naturel oblige RAINS à distinguer les expressions naturelles des entités ayant une sémantique formelle. Cette distinction a comme conséquence le fait que la traduction d'une règle RAINS de l'anglais vers une langue dont la construction des phrases est similaire à l'anglais est immédiate. Nous avons illustré ce propos à la section 4.2 en page 89.

4.8 Conclusion du chapitre

Notre but est de proposer un langage contrôlé compatible à la non-naturalité des libellés des entités IFC, permettant d'écrire des règles devant servir à la détection de non-conformités des maquettes numériques par rapport à la réglementation . De plus, les experts métiers, qui sont les destinataires de ce langage, doivent avoir l'assurance que les exigences qu'ils rédigent avec ce langage peuvent être transformées sous une forme exécutable, sans supposition et sans heuristique. Pour cela, nous avons proposé un nouveau langage contrôlé, nommé RAINS, qui répond à ces trois exigences.

RAINS est un langage contrôlé qui dispose d'une syntaxe formelle, que nous avons représenté à l'aide d'une grammaire non-contextuelle et encodée à l'aide de la notation *Extended Backus-Naur Form*. La syntaxe de RAINS nous permet de voir que chaque exigence RAINS se présente sous la forme d'une règle et décrite éventuellement par des annotations. Cette règle de non-conformité se présente sous une forme mettant en évidence l'antécédent (partie ***si***) et le conséquent (partie ***alors***) de la règle. Par ailleurs, pour améliorer la compréhension des exigences, RAINS nécessite que l'antécédent et le conséquent des règles soient formés d'une conjonction d'assertions atomiques. Chacune de ces assertions doit nécessairement être de l'un des types d'assertions prédéfinis du langage RAINS (voir section 4.2). En outre, quel que soit le type d'assertion, elle se compose d'un ensemble d'entités RAINS, délimitées par des guillemets ("). Ces entités sont soit des termes de l'ontologie conceptualisant le domaine (classes, propriétés, individus), soit des prédicats prédéfinis (par exemple des fonctions de comparaison), soit

des marques de négation ou de quantification universelle. Aussi, RAINS permet dans une assertion de faire usage de manière libre, d'expressions en langage naturel. Cela permet, si c'est jugé nécessaire, de lier les entités de manière à obtenir une phrase se rapprochant le plus possible du langage naturel, lisible et compréhensible. Enfin, nous avons souligné le fait que, bien que la syntaxe de RAINS utilise des termes de langue anglaise (`If`, `Then`, `not`, etc.), la traduction de ces termes et donc celle des exigences RAINS en d'autres langues que l'Anglais est immédiate. De plus, relevons que RAINS est un langage doté d'une sémantique formelle. Chaque phrase RAINS est équivalente à une requête SPARQL.

Par ailleurs, nous avons repris les éléments de classification proposés par Kuhn [2014], pour caractériser RAINS. Nous avons vu que RAINS est un langage de type (F W D A). En d'autres termes il permet de faciliter une représentation formelle du langage, il est destiné à l'écriture, il est inspiré d'un domaine précis (celui de la Construction) et est partie d'une initiative académique. Toutefois notons que RAINS peut être utilisé pour écrire des règles de non-conformité dans tout domaine conceptualisé à l'aide d'une ontologie. En outre, la position de RAINS dans l'espace PENS est donnée par les coordonnées (P^5, E^3, N^3, S^3). Autrement dit, RAINS est un langage précis dont la formalisation ne nécessite ni hypothèse, ni heuristique, et dont l'expressivité est "moyenne". Aussi, RAINS est un langage avec une prépondérance d'éléments naturels mais n'utilisant pas de phrases naturelles et qui dispose d'une description longue (plus de 10 pages).

Enfin, nous avons illustré les différences syntaxiques entre RAINS, PENG-D et PNL sur la base de quelques exemples de règles. Nous avons pu constater que l'ensemble des mots réservés de RAINS est réduit par rapport à celui de PENG-D et PNL. Ces derniers emploient en plus des délimiteurs de règles (`if`, `then`, etc.), et autres prédicats de comparaisons, et marqueurs de négation et de restriction universelle, d'autres mots réservés tels que les déterminants, des auxiliaires et des pronoms. En outre, nous avons pu mettre en évidence le fait que dans certains cas, une règle écrite en PENG-D ou PNL pouvait déboucher sur une perte des informations descriptives mentionnées dans la règle originale, alors qu'avec RAINS, ces descriptions peuvent être exprimées à l'aide d'annotations. Aussi, contrairement à PNL et PENG-D, RAINS propose un mécanisme de chainage de propriétés qui permet de réduire la verbosité des exigences formelles. Pour finir, nous avons mis en exergue l'importance de l'utilisation libre des expressions en LN dans les règles RAINS. Ce mécanisme permet d'ajouter des commentaires et précisions utiles dans les règles, d'améliorer si nécessaire la naturalité des règles ou encore de faciliter la traduction des règles RAINS, de l'anglais vers d'autres langues, et vice versa.

Maintenant que nous disposons d'un langage contrôlé pour permettre un dialogue sans

intermédiaire entre les experts métiers et l'ordinateur, nous nous proposons d'assister les experts dans la formalisation de corpus technico-réglementaires à partir de normes en langage naturel. Pour cela, nous proposons un service permettant la formalisation automatique de règles métiers. Ce service prend en entrée une exigence dans sa forme naturelle, ainsi que l'ontologie de domaine concernée, et retourne une version RAINS de celle-ci.

Chapitre 5

FORSA : une approche de formalisation automatique de règles métiers

Contents

Au chapitre précédent, nous avons présenté le langage contrôlé RAINS. Ce langage permet aux experts-métiers d'exprimer les exigences réglementaires sans plus recourir à des spécialistes en représentation de connaissances. RAINS permet donc à l'expert métier, qui comprend les réglementations et qui maîtrise la conceptualisation du domaine, de pouvoir constituer un dépôt de règles formelles, en totale autonomie. Ce dépôt de règles servira à alimenter les opérations de contrôles automatiques de la conformité des produits d'ingénierie, encore sous forme de maquette numérique, vis-à-vis des normes en vigueur.

Cependant, la réécriture des normes, originalement en langage naturel, en des règles RAINS demeure une opération entièrement manuelle. Au regard de la taille du corpus technico-réglementaire (voir section 2.1.2), l'automatisation, même partielle, de ce processus de rédaction est souhaitable et souhaitée par les experts métiers.

C'est dans cette optique que nous proposons un service de formalisation automatique de règles métiers. Ce service implémente une approche de formalisation dénommée FORSA. FORSA permet de transformer une règle (sous la forme d'une phrase), initialement en langage naturel (cf. section 3.1.5), en une règle RAINS ***valide***. Rappelons qu'une règle RAINS est valide si sa syntaxe est correcte (c'est-à-dire qu'elle a un *logical pattern* explicite, qu'elle utilise des entités issues de l'ontologie de domaine ou des mots clés du langage, et que l'agencement de ses entités est correcte - voir *listing* 4.1) et si dans chaque annotation et assertion les contraintes de validité, au regard de l'ontologie de domaine, sont respectées (voir section 4.5). Ainsi, les mécanismes employés dans FORSA supposent l'existence d'une ontologie décrivant le domaine de connaissance duquel est issue la règle à formaliser.

FORSA est une approche basée sur des ***heuristiques*** et utilisant des outils de base du ***traitement automatique du langage naturel*** [1] (TALN). Les différentes étapes de FORSA sont présentées par la figure 5.1. On peut ainsi noter que FORSA comprend quatre (04) principales étapes qui à leur tour se décomposent en des sous-opérations.

Étape 1 Une décomposition de la règle en des assertions atomiques (mais encore en langage naturel). Une règle RAINS se composant d'un ensemble d'assertions semi-formelles, cette étape permet déjà d'entrevoir notre règle comme un ensemble de faits simples (voir section 5.1)

Étape 2 La transformation de chacune des assertions de la règle, en assertions RAINS (voir section 5.2)

Étape 3 La formation de l'antécédent de la règle par l'assemblage des assertions RAINS (voir section 5.3)

Étape 4 La détermination du conséquent de la règle (voir section 5.4)

1. En Anglais : *Natural Language Processing* (NLP)

Notons qu'une première ébauche de cette approche est présentée dans l'article [Kacfah Emani *et al.* 2016a].

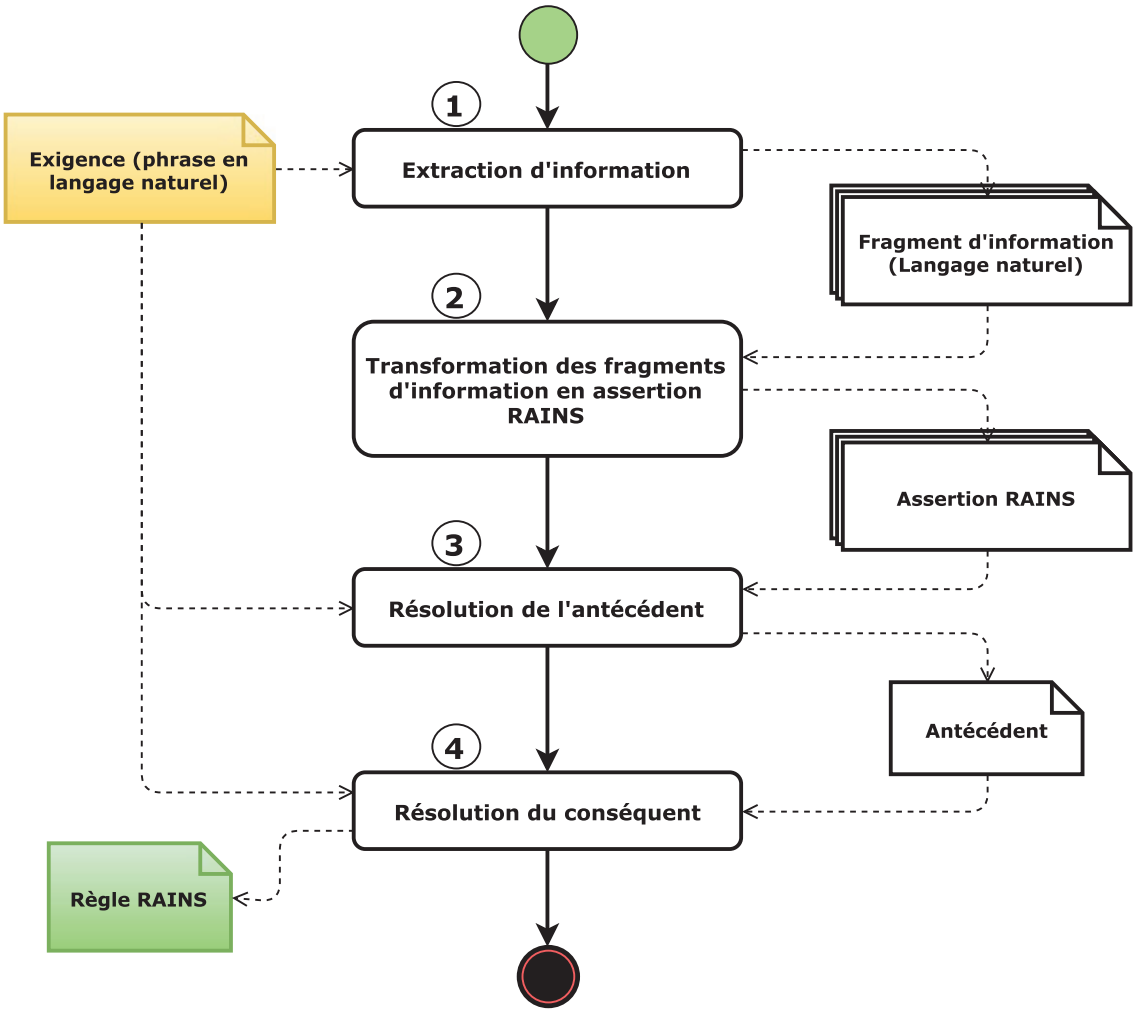


FIGURE 5.1 – Différentes étapes de l'approche de formalisation automatique de règles FORSA.

Dans les sections qui suivent, nous présentons en détail les étapes de FORSA. Nous illustrons chacune de ses étapes successives en prenant pour entrées l'exigence ***"The rated speed of the lifting platform shall not be greater than 0,15 m/s"*** ainsi que l'ontologie IfcOWL (voir figure 4.1).

5.1 Identification des assertions

Le but de l'approche FORSA est de transformer une exigence réglementaire en une règle RAINS valide. Comme nous l'avons vu, une règle RAINS se compose d'annotations, d'un antécédent et d'un conséquent, tous construits autour d'assertions. Dans cette première étape, nous identifions les assertions, encore en langage naturel, qui composent

l'exigence.

Dans le domaine du traitement automatique du langage naturel (TALN), cette opération porte le nom d'*Open Information Extraction* (OIE), parfois appelée en français ***extraction d'informations ouverte***. L'OIE a été introduite pour la première fois par Banko *et al.* [2007]. Elle consiste à extraire d'une phrase, l'ensemble des informations qui la composent. Généralement, les résultats sont retournés sous la forme d'une liste de triplets ⟨`sujet, verbe/prédicat, objet`⟩. Plusieurs outils d'OIE sont proposés dans la littérature. Nous pouvons les classifier en deux grandes familles :

1. Les outils ayant recours à l'***apprentissage*** (*machine learning*). Dans cette famille, on retrouve des outils tels que TEXTRUNNER [Banko *et al.* 2007], REVERB Fader *et al.* [2011] ou encore OLLIE [Maussam *et al.* 2012]. Tous ces outils nécessitent l'existence d'un corpus annoté, comprenant plusieurs phrases, à partir duquel ils apprendront à découvrir les relations entre les termes, ce qui les permettra d'identifier les informations qui composent une phrase.
2. Les outils d'extraction basés sur les ***heuristiques***. Dans ce second groupe, l'existence d'un corpus d'apprentissage n'est pas nécessaire. Les approches implémentées par les outils se basent uniquement sur des heuristiques pour dériver les informations, à partir des constituants de la phrase ou des dépendances syntaxiques entre les mots de la phrase. C'est le cas des outils tels que Claus-IE [Del Corro & Gemulla 2013] ou encore CSD-IE Bast & Haussmann [2013].

Les outils d'extraction d'informations basés sur les heuristiques sont les plus récents et obtiennent de meilleurs scores sur les corpus d'évaluation régulièrement utilisés par les chercheurs travaillant sur les problématiques de l'OIE (pour une comparaison détaillée voir [Bast & Haussmann 2013, section 4]). De plus, les outils de cette catégorie ne requièrent aucun travail supplémentaire comme c'est le cas avec la mise sur pied de corpus annotés pour entraîner les outils de *machine learning*. Pour ces raisons, pour l'implémentation de FORSA, nous avons porté notre choix sur les outils d'OIE de cette catégorie.

De l'analyse que nous avons fait des outils Claus-IE (*Clause-based IE*) développé par Del Corro & Gemulla et CSD-IE (*Contextual Sentence Decomposition for IE*) proposé par Bast & Haussmann, ce dernier est plus adapté à nos besoins. En effet, les informations extraites en utilisant l'approche de CSD-IE répondent à des aspects de qualité qui rendent ces informations plus pertinentes pour FORSA. Ces aspects de qualités (*quality aspects* - [Bast & Haussmann 2013, section 3.A]) sont :

1. ***La précision des faits extraits.*** Lorsqu'un outil d'extraction d'informations effectue le traitement d'une phrase, il peut en ressortir des faits erronés. Ces erreurs peuvent, par exemple, provenir de ce que ces faits ne soient pas exprimés dans

la phrase d'entrée (par exemple ⟨`the rated speed, is a, lifting platform`⟩), ou de ce que ces faits soient à proprement parler "dénués de sens" (par exemple ⟨`greater, than, 0.15 m/s`⟩). Dans la conception de l'approche CSD-IE, les auteurs proposent des heuristiques pour éviter d'obtenir des informations "dénuées de sens". Pour illustration, une de ces heuristiques veut que le prédicat d'un triplet soit nécessairement ***basé sur un verbe***.

2. ***La couverture.*** Toute approche d'OIE essaye d'extraire d'une phrase donnée, le maximum de triplets. En comparant les informations effectivement présentes dans la phrase originale et celles disponibles dans les triplets-résultats, il peut arriver que ces derniers manquent de refléter toutes les informations de la phrase d'entrée. Dans l'approche CSD-IE, les auteurs essayent de garantir la couverture en s'assurant que chaque terme de la phrase originale figure dans au moins un triplet.

3. ***La minimalité.*** Qu'un triplet soit correct (c'est-à-dire l'information qu'il véhicule se retrouve en intégralité dans la phrase d'entrée) n'est pas une condition suffisante pour qu'il soit pertinent pour tout processus devant utilisé ce triplet. En effet, il arrive parfois qu'on puisse encore décomposer un triplet (qu'on peut envisager comme une sous-phrase), en des triplets plus petit. Par exemple, partant de la phrase *"A lifting platform shall have a length that is less than 1400 mm and a height that is above 2 m"*, on peut obtenir le triplet (τ_α) ⟨`A lifting platform, shall have, a length that is less than 1400 mm and a height that is above 2 m`⟩. Nous pouvons constater que de cet unique triplet, on peut obtenir les deux triplets suivants : (τ_a) ⟨`A lifting platform, shall have, a length that is less than 1400 mm`⟩ et (τ_b) ⟨`A lifting platform, shall have, a height that is above 2 m`⟩. Nous disons de (τ_α) qu'il n'est pas minimal et de (τ_a) et (τ_b) qu'ils sont minimaux ou atomiques. La minimalité est une des conditions à respecter par les triplets produits par CSD-IE.

Disposer de triplets atomiques informatifs et couvrant l'ensemble des informations de la phrase d'entrée nous permet de nous rapprocher de la syntaxe de RAINS. Ces principes sur lesquels repose CSD-IE, ne se retrouvent pas dans l'outil Claus-IE.

Un passage de notre exemple d'exigence par CSD-IE, nous fourni un unique triplet : (τ_0)⟨`The rated speed of the lifting platform, shall not be, greater than 0,15 m/s`⟩.

5.2 Transformation des assertions du langage naturel en assertions RAINS

Maintenant que nous avons décomposé notre exigence d'entrée en un ensemble d'assertions atomiques, nous allons transformer ces assertions, encore en langage naturel (***notation : assertion-LN***), en des assertions respectant les spécifications du langage contrôlé RAINS. Les étapes de cette transformation telle que nous les présentons dans cette section sont effectuées pour chaque assertion-LN prise individuellement (voir figure 5.2).

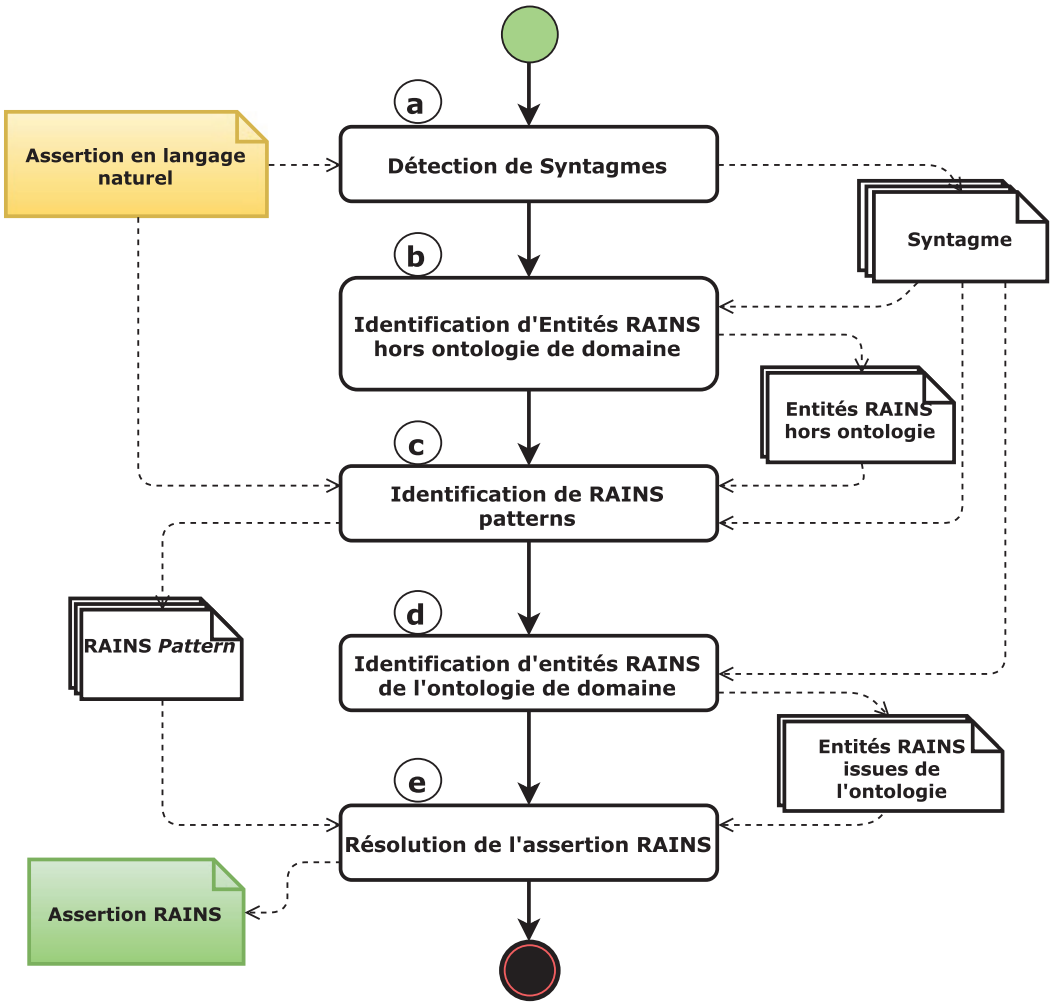


FIGURE 5.2 – Étapes de transformation d'une assertion en langage naturel en une assertion RAINS.

5.2.1 Détection des syntagmes

Chaque assertion RAINS se compose d'entités. À ce stade, nous nous posons la question de savoir si dans notre assertion-NL, on peut déjà identifier des groupes de mots sus-

ceptibles de devenir ou de produire des entités RAINS. En linguistique, ces groupes de mots sont appelés des ***syntagmes***. Un syntagme se définit comme un mot ou une combinaison de mots qui se ***suivent*** et qui produisent un ***sens***. Dans les langues française et anglaise, il existe principalement cinq types de syntagmes [2] : ***adjectivaux*** (ADJP), ***adverbiaux*** (ADVP), ***nomimaux*** (NP), ***prépositionnels*** (PP) et ***verbaux*** (VP). Comme leur nom l'indique, ils sont construits, respectivement, autour d'un adjectif, d'un adverbe, d'un nom, d'une préposition ou d'un verbe.
Plusieurs outils de détection de syntagmes sont disponibles dans la communauté du TALN. Pour l'implémentation de cette opération dans notre approche, nous avons utilisé l'*Illinois Chunker* de Punyakanok & Roth [2001]. Pour la sous-phrase dénotée par l'unique assertion, (τ_0), de notre exigence réglementaire, avec l'*Illinois Chunker*, nous obtenons les syntagmes suivants (***dans cet ordre***) :

1. **NP** (DT The) (VBN rated) (NN speed)
2. **PP** (IN of)
3. **NP** (DT the) (VBG lifting) (NN platform)
4. **VP** (MD shall) (RB not) (VB be)
5. **NP** (JJR greater) (IN than) (CD 0,15) (NN m) (NN /)
6. **VP** (VBZ s)

On peut constater une erreur dans cette identification des syntagmes. En l'effet l'abréviation *m/s* est faussement découpée par l'I*llinois chunker*. Nous présentons l'impact des erreurs dues aux outils que nous utilisons à la section 6.3.4.
En plus des syntagmes (et de leur type), les mots qui les composent possèdent une ***étiquette morpho-syntaxique*** : **DT** (déterminant), **VBN** (verbe au participe passé), **IN** (préposition), **VBG** (verbe au participe présent), **NN** (non singulier), **VB** (verbe à l'infinitif), **MD** (auxiliaire modal), **RB** (adverbe), **JJR** (adjectif au comparatif) et **VBZ** (verbe à la troisième personne du singulier). Pour la liste complète des étiquettes morpho-syntaxiques, bien vouloir se reporter à l'annexe A.

5.2.2 Détection des entités hors ontologie de domaine

Lorsque nous avons présenté la syntaxe du langage RAINS (cf. section 4.3), nous avons mentionné qu'une phrase RAINS se construit autour d'assertions, elles-mêmes se composant d'entités (***notation : entité RAINS***). Aussi, nous avons relevé le fait que les entités RAINS étaient de deux principaux types. Une entité RAINS peut faire référence à un concept, ou une propriété ou un individu de l'ontologie de domaine servant de vocabulaire aux phrases RAINS ou alors une entité RAINS peut être un terme général tel

2. Les abréviations (ADVP, ADJP, NP, PP, VP) sont reprises du Penn Treebank. Le *P* signifie *phrase* [Marcus *et al.* 1993]. Ces abréviations sont quasiment devenues des standards car repris dans la plupart des outils de TALN.

qu'on en retrouve dans les phrases naturelles pour exprimer la négation, la comparaison, les quantités, etc. C'est ce second type d'entités RAINS, c'est-à-dire hors ontologie de domaine, que nous détectons à cette étape.
Pour cela, dans chaque syntagme, on recherche la présence des termes reflétant des entités RAINS hors ontologie de domaine. Rappelons que la liste de ces termes est soit connue (fonctions de comparaisons, marque de négation, marque de restriction universelle), soit disponible sous forme d'expressions régulières (pour la détection des entiers, des flottants, des dates, etc.) (voir la ligne signalée par ❸ et les lignes suivantes du *listing* 4.1). Chacun de ces termes identifiés est d'un type précis et à ce type correspond une ***lettre de code*** telle que présentée dans le tableau 5.1. Ces codes permettent de définir ce que nous appelons ***configurations*** pour nos assertions.

Tableau 5.1 – Lettre de code des entités RAINS

Type de l'entité RAINS	Code
`Annotation`	**I**
`Concept`	**C**
`DatatypeProperty`	**D**
`ChainOfObjectProperties`	**O**
`ObjectProperty`	**O**
`BooleanDatatypeProperty`	**B**
`Individual`	**I**
`Comparator`	**K**
`Neg`	**N**
`UniversalRestrictionKeyword`	**U**
`Integer`	**i**
`Float`	**F**
`Boolean`	**b**
`String`	**S**
`Date`	**d**

Au regard des syntagmes obtenus à l'étape précédente, on peut détecter les entités RAINS indiquées ci-dessous :

1. **NP** (DT The) (VBN rated) (NN speed)
2. **PP** (IN of)
3. **NP** (DT the) (VBG lifting) (NN platform)
4. **VP** (MD shall) not_{N} (VB be)
5. **NP** $\text{greater than}_{\text{K}}$ $0{,}15_{\text{F}}$ (NN m) (NN /)
6. **VP** (VBZ s)

Au sortir de cette étape, nous disposons des entités RAINS hors ontologie ainsi que d'***un fragment de la configuration*** de notre future assertion RAINS. Ce fragment est dénoté par la variable `seedPattern` dans nos algorithmes. Si à ce stade nous ne détectons pas

d'entités RAINS, cela veut dire que l'assertion RAINS que nous obtiendrons à la fin de la transformation de l'assertion-LN en une assertion RAINS, ne fera pas usage d'entités RAINS hors ontologie de domaine.
Pour notre exemple, les trois entités RAINS déjà identifiées sont : `not`, `greater than` et `0.15`. Le fragment de configuration de notre future assertion RAINS est ***N*KF***. Le symbole * signale des emplacements à occuper par les lettres de codes [3], des entités RAINS issues de l'ontologie de domaine et que nous allons identifier au cours des étapes suivantes.

5.2.3 Identification des configurations RAINS candidates

L'un des résultats obtenus à l'étape précédente est un fragment de la configuration finale de notre assertion RAINS. Le but de cette étape est d'obtenir, la liste de toutes les configurations des assertions RAINS incluant ce fragment. Pour obtenir cette liste, nous allons supprimer de l***a liste de toutes les configurations RAINS possibles*** celles qui ne contiennent pas ce fragment. Mentionnons que la liste exhaustive de ces configurations est fournie par les expressions, en notation EBNF, des différentes assertions RAINS (voir *listing* 4.1). Pour illustration, prenons par exemple l'expression d'une :

— *Annotation*
La configuration d'une annotation est le symbole **I** [4]

— *Single concept assertion*
`Text?, Concept, Text?`
Elle induit une seule configuration possible : **C**

— *Double concept assertion*
`Text?, Neg?, Text?, Concept, Text?`
Deux configurations sont possibles dans ce cas : **CC** et **CNC**

— *Inverted data assertion*
`Text?, DatatypeProperty, Text?, Concept, Text?, Neg?, Comparator, Quantity`
Le nombre de configurations dans ce cas est plus grand (voir la définition du symbole `Quantity` dans le *listing* 4.1, page 80) : **DCKi**, **DCKF**, **DCKb**, **DCKS**, **DCKd**, **DCKDC**, **DCNKi**, **DCNKF**, **DCNKb**, **DCNKS**, **DCNKd** et **DCNKDC**.

Notons que le symbole `Text`, signifiant l'utilisation possible de textes en langage naturel, n'a aucune incidence sur les configurations des assertions. En effet, dans les configurations figurent uniquement les codes des entités RAINS. Or le symbole `Text` ne désigne pas une entité RAINS.
Comme le montre la liste des exemples ci-dessus, nous pouvons disposer de l'ensemble des configurations couvertes par les assertions RAINS. Pour sélectionner, parmi l'ensemble des configurations possibles (dénoté par la variable `RAINS_PATTERNS`), les configu-

3. Une * peut être remplacée par ***zéro ou plusieurs*** lettres de codes.
4. Ce symbole est le même que celui d'une entité RAINS faisant référence à un individu (c'est-à-dire `Individual`). Pour des explications relatives à ce choix, bien vouloir se reporter à la section 5.2.5 en page 131.

rations candidates compatibles avec le fragment de configuration (`seedPattern`) obtenu à l'étape précédente, nous utilisons un filtre. Ce filtre est opéré au travers de l'algorithme présenté par le *listing* 5.1.
Dans ce pseudo-code, il y a deux points centraux :

1. À la ligne ①, on marque le fait qu'une configuration RAINS (`rainsPattern`) n'est candidate que si la chaîne de caractères qui la représente contient le motif décrit par le fragment de configuration (`seedPattern`), obtenu à l'étape précédente (section 5.2.2)
2. À la ligne ②, on soumet `rainsPattern` à une seconde condition. Vu que `seedPattern` est le reflet de toutes les entités RAINS hors ontologie et qui sont contenues dans l'assertion, alors `rainsPattern` ne peut pas contenir ***d'autres*** entités hors ontologie en dehors de celles du seed pattern. Cette condition est signifiée à la ligne ③.

Listing 5.1 – Pseudo (java) code pour l'identification des configurations possibles d'une assertion

```
1  private static HashSet<String> getPatterns2(String seed){
     HashSet<String> nonOntologyEntitiesCodes = new HashSet<>();
3    nonOntologyEntitiesCodes.add("K");
     nonOntologyEntitiesCodes.add("N");
5    nonOntologyEntitiesCodes.add("U");
     nonOntologyEntitiesCodes.add("i");
7    nonOntologyEntitiesCodes.add("F");
     nonOntologyEntitiesCodes.add("b");
9    nonOntologyEntitiesCodes.add("S");
     nonOntologyEntitiesCodes.add("d");
11
     HashSet<String> possiblePatterns = new HashSet<>();
13   for (int i = 0; i < seed.length(); i++) {
         nonOntologyEntitiesCodes.remove(Character.toString(seed.charAt(i)));
15   }
     for(String rainsPattern:RAINS_PATTERNS){
17       Pattern p = Pattern.compile(seed);
         Matcher m = p.matcher(rainsPattern);
19     ①  if(m.find()){// Si rainsPattern contient la configuration de
           seedPattern
               boolean keep = true;
21             for(String entityCode:nonOntologyEntitiesCodes){
       ②       keep = keep & !rainsPattern.contains(entityCode); }
23         //seedPattern ne contient que les codes d'entités non ontologiques
           //On conserve un rainsPattern s'il ne contient pas d'autres entités non
               ontologiques
25         ③   if(keep){ possiblePatterns.add(rainsPattern); }
       } // fin if(m.find())
27   } // fin for(String rainsPattern:RAINS_PATTERNS)
     return possiblePatterns;
29 }
```

Pour cette opération d'identification des configurations RAINS, il existe deux cas particuliers.

> ***Cas 1.*** Aucune entité non ontologique n'a été identifiée et donc la variable `seedPattern`, dans le *listing* 5.1, a la valeur * qui symbolise 0 ou plusieurs lettres de code. La liste des configurations RAINS correspondant à ce pattern est **I**, **BC**, **CC**, **C**, **COC**, **CB**, **COB** et **COI**. Elle comprend tous les patterns RAINS qui ***n'utilisent pas d'entités non-ontologiques***.
> ***Cas 2.*** Aucun *pattern* autorisé par RAINS ne correspond au motif en entrée. Dans ce cas, ***nous considérons que l'assertion RAINS qui découlera de cette assertion-LN sera erronée et donc nous interrompons la semi-formalisation de cette assertion-LN***.

Si l'on applique cet algorithme à notre exemple, dont le motif (`seedPattern`) est ***N*KF***, nous obtenons la liste des configurations suivantes : **CDNKF**, **DCNKF**, **CODNKF**. Ces configurations nous suggèrent de rechercher un concept, une *datatype property*, et éventuellement une *object property* pour obtenir une version RAINS de notre assertion en LN. À la section suivante, nous nous attardons sur l'identification de ces entités.

5.2.4 Identification d'entités de l'ontologie de domaine et détermination des assertions RAINS candidates

À la section 5.2.2 nous avons identifié les entités hors ontologie de domaine, qui composent notre assertion. Rendu à cette étape, notre but est d'obtenir les entités, cette fois ci issues de l'ontologie de domaine, qui rentrent dans notre assertion. Rappelons que ces entités peuvent être de quatre (04) types : Concept (**C**), Individu ou annotation (**I**), *datatype property* (**D**), *object property* (**O**). Aussi, rappelons qu'à cette étape, nous disposons des syntagmes qui composent notre assertion-LN (voir section 5.2.1). Nous décrivons donc comment à partir de ces syntagmes, et des libellés des termes de l'ontologie de domaine, nous obtenons les entités ontologiques devant permettre de formaliser notre assertion.

Pour cela, nous considérons, ***heuristiquement***, que les mots devant nous permettre d'identifier les termes ontologiques, se trouvent dans des syntagmes adjectivaux (ADJP), nominaux (NP) ou verbaux (VP). Ainsi, nous considérons que les syntagmes adverbiaux, prépositionnels et tous les autres types de syntagmes ne sont pas sémantiquement pertinent dans ce processus. Dans les syntagmes adjectivaux, nominaux et verbaux, tous les mots qu'on y trouve ne sont pas d'égale utilité à la réalisation de notre tâche. Pour ne conserver que les mots que nous considérons utiles, nous procédons à un double filtrage que constituent les ***élagages syntaxique*** et ***sémantique***.

A. Élagage syntaxique

Dans cette étape de filtrage, pour chaque syntagme de type adjectival, nominal ou verbal, seuls les ***noms*** (dont les étiquettes morpho-syntaxiques sont **NN**, **NNS**, **NNP** ou **NNPS**), les ***verbes*** (**VB**, **VBD**, **VBG**, **VBN**, **VBZ**) et les ***adjectifs*** (**JJ**, **JJR**, **JJS**) sont utilisés. Cela veut dire que nous écartons les déterminants (**DT**), les prépositions (**IN**), les adverbes (**RB**, **RBR**, **RBS**), etc. qui pourraient se trouver dans ces syntagmes.

B. Élagage sémantique

Après avoir effectué l'élagage syntaxique, nous procédons à un second tamis. Cette fois-ci, nous éliminons, heuristiquement, les mots qui sémantiquement ne contiennent aucune informations. Parmi ces mots, on retrouve :

- Les ***auxiliaires***. Ce sont ***être*** (*be*), ***avoir*** (*have*), *do*
- Les ***auxillaires modaux***. C'est le cas des verbes tels que pouvoir (*can*, *may*), devoir (*must*, *shall*, *ought*), vouloir (*will*), etc.
- Les ***mots hors vocabulaire***. Rappelons que la formalisation des normes, telle qu'envisagée dans notre travail, suppose l'existence d'une ontologie de domaine. Les libellés (en anglais *labels*) des termes de cette ontologie, un terme pouvant disposer d'un ou plusieurs libellés synonymes, sont considérés comme étant le vocabulaire du domaine de connaissances que ces normes encadrent. Dans notre domaine d'aplication, le vocabulaire se compose des termes de l'ontologie IfcOWL enrichie des termes relatifs aux plateformes élévatrices (voir figure 4.1). Nous qualifions un mot de l'exigence réglementaire de "mot hors vocabulaire", si sa ***racine*** (en anglais *stem*) ne se retrouve pas dans l'ensemble des racines des mots du vocabulaire du domaine.

Si nous appliquons les filtres syntaxique et sémantique aux syntagmes de notre exemple, nous obtenons :

1. **NP** (~~DT The~~) (VBN rated) (NN speed) **`#Élagage syntaxique`**
2. ~~**PP (IN of)**~~ **`#Type de syntagme non considéré`**
3. **NP** (~~DT the~~) (VBG lifting) (NN platform) **`#Élagage syntaxique`**
4. **VP** (MD ~~shall~~) not$_{N}$ (VB ~~be~~) **`#Élagage syntaxique`**
5. **NP** greater than$_{K}$ 0,15$_{F}$ (NN ~~m~~) (NN ~~/~~) **`#Élagage sémantique`**
6. **VP** (VBZ ~~s~~) **`#Élagage sémantique`**

C. Identification des entités de l'ontologie de domaine proprement dites

Pour chaque syntagme éligible, et débarrassé des mots écartés par les élagages syntaxique et sémantique et aussi ***des mots ayant servis à l'identification des entités hors ontologies*** (voir section 5.2.2), nous procédons comme suit :

1. Si le syntagme est ***adjectival*** et ***non vide*** (c'est-à-dire qu'il reste des mots après les filtres sus-mentionnés), alors :
   On recherche les ***top-n*** propriétés d'une part et les ***top-n*** concepts et/ou individus d'autre part, ayant la plus grande similarité (en anglais *string similarity*) avec le syntagme.
2. Si le syntagme est ***verbal*** et
   (a) ***Non-vide*** : on recherche les ***top-n*** propriétés (*datatype* et *object properties*) ayant la plus grande similarité avec le syntagme
   (b) ***Vide*** : On considère que ce syntagme verbal désigne une éventuelle entité de type *object property*. Selon les cas (que nous présentons plus loin), on décidera de rejeter définitivement cette propriété ou alors de l'exploiter. On représentera cette entité par le libellé `undetermined property` et la lettre de code **X**.
3. Si le syntagme est ***nominal***, ***non vide*** et ***possède au moins un nom*** [5], alors :
   pour ***chaque sous-syntagme nominal*** de ce syntagme, on recherche les ***top-n*** propriétés d'une part et les ***top-n*** concepts et/ou individus d'autre part, ayant la plus grande similarité avec ce sous-syntagme.

Il est important de noter que chaque fois que nous calculons la similarité entre un terme de l'assertion-LN et un terme de l'ontologie de domaine, nous conservons ce score de similarité. Cela nous permettra d'attribuer un score global à une assertion RAINS, sur la base du score de similarité des entités ontologiques présentes dans cette assertion (voir algorithme B.1, page 190).

Similarité entre chaîne de caractères. Nous constatons que le processus de recherche des entités de l'ontologie les plus adéquates pour la formalisation de notre assertion, fait appel à des fonctions de calcul de similarité entre chaîne de caractères. Pour l'implémentation de FORSA, nous avons utilisé la ***moyenne***, entre la ***similarité cosinus*** [Garcia 2006] et de la ***similarité Jaccard_2*** [Nguyen *et al.* 2013]. La première nous permet de regarder les deux chaînes de caractères à comparer comme étant des ensembles de mots. Cela permet de mesurer la proximité des deux chaines de caractères, mots à mots, sans tenir compte de l'ordre des mots dans les chaînes de caractères. Quant à la mesure Jaccard_2, elle permet de comparer les chaînes de caractères à un niveau plus fin que la mesure cosinus, c'est-à-dire caractères par caractères [6]. Aussi, elle permet de tenir compte de l'ordre des caractères et des mots.
Enfin, mentionnons que la comparaison des chaînes de caractères s'effectue après que ces dernières aient été ***lemmatisées*** (en anglais *lemmatized*). La lemmatisation est une

5. Bien qu'un syntagme nominal soit construit autour d'au moins un nom, il se peut que l'élagage sémantique vide le syntagme de tous ses noms.

6. Jaccard_2 est une adaptation de la mesure Jaccard [Hadjieleftheriou & Srivastava 2010]. À la base, Jaccard compare les chaines de caractères en prenant comme unité le mot. Jaccard_n adapte Jaccard pour effectuer la comparaison par groupe de n caractères [Nguyen *et al.* 2013]. Dans notre cas, on a $n = 2$.

opération de base du TALN qui permet d'obtenir la forme ***canonique*** des mots. Autrement dit, c'est une opération qui débarasse les mots de toutes les inflexions telles que les marques de ***genre*** (masculin, féminin), de ***pluriel***, de ***conjugaison***, etc. Ainsi, la lemmatisation d'un verbe est la forme à l'infinitif de ce verbe, celle d'un nom, sa forme au masculin singulier.

Réduction de l'espace de recherche. En exécutant les étapes décrites pour l'identification des entités pour chaque syntagme, on constate que l'on obtient plusieurs groupes d'entités pour une assertion. En effet, une assertion donne droit à un ***maximum*** de

$$\mathbf{2.n.|A|} \times \mathbf{n.|V|} \times \mathbf{2.n.|N_S|} = \mathbf{4.n^3.|A|.|V|.|N_S|}$$

possibilités d'instanciations, où $|A|$ est le nombre de syntagmes adjectivaux, $|V|$ le nombre de syntagmes verbaux et $|N_S|$ le nombre de sous-syntagmes nominaux de l'assertion. Ce nombre de possibilités peut très vite devenir important pour une phrase comportant plusieurs syntagmes et pour une grande valeur de n. Rappelons que n désigne, pour chaque terme de l'assertion-LN, le nombre d'entités candidates à la formalisation de ce terme.
Pour réduire ce nombre, nous avons fixé un seuil de similarité minimale, lors du calcul de la similarité entre les termes des syntagmes et ceux de l'ontologie. Ainsi un terme de l'ontologie ne sera candidat à la formalisation d'une assertion que si sa similarité avec les mots du syntagme est supérieure à une valeur σ. On passe ainsi pour chaque syntagme des n meilleurs candidats pour le représenter, aux $\boldsymbol{n'}$ meilleurs candidats, avec $\mathbf{0} \leq \mathbf{n'} \leq \mathbf{n}$. Un syntagme pour lequel $n' = 0$ sera considéré comme sémantiquement vide et ne sera pas pris en compte dans la formalisation.

Pour notre exemple, nous obtenons les résultats suivants ($n = 3$ et $\sigma = 0.4$ - ces valeurs sont discutées à la section 6.3.4 page 176)

1. **NP** (~~DT The~~) (VBN rated) (NN speed)
   Un seul sous-syntagme nominal dont le libellé est "rated speed"
   Entités candidates = {:ratedSpeed$_\mathbf{D}$}
2. **NP** (~~DT the~~) (VBG lifting) (NN platform)
   Un seul sous-syntagme nominal dont le libellé est "lifting platform"
   Entités candidates = {:LiftingPlatform$_\mathbf{C}$, :LIFTINGGEAR$_\mathbf{I}$}
3. **VP** (MD ~~shall~~) not$_\mathbf{N}$ (VB ~~be~~)
   Ce syntagme verbal étant vide (tous les mots étant élagués), il est considéré comme une potentielle *object property*
   Entités candidates = {:undeterminedProperty$_\mathbf{X}$}
4. **NP** greater than$_\mathbf{K}$ 0,15$_\mathbf{F}$ (NN ~~m~~) (NN ~~/~~)
   Syntagme nominal vide, donc pas d'entité candidate.

5. **VP** (VBZ s)
   Ce syntagme verbal vide
   Entités candidates = {:undeterminedProperty$_X$}

La liste des assertions RAINS candidates pour notre assertion en LN est donc :

1. (ρ_0) "rated speed" "Lifting platform" "not" "undetermined property" "greater than" "0.15" "undetermined property"
   Configuration : **DCNXKFX**
2. (ρ_1) "rated speed" "LIFTING_GEAR" "not" "undetermined property" "greater than" "0.15" "undetermined property"
   Configuration : **DINXKFX**

5.2.5 Résolution de l'assertion RAINS

Au terme de l'étape précédente, nous obtenons une liste d'assertions RAINS candidates. Il se pose donc la question de la détermination de l'assertion RAINS finale, en utilisant l'une de ces candidates. Pour effectuer ce choix, nous allons classer les assertions RAINS candidates dans trois catégories disjointes. Avant de présenter ces catégories, rappelons que :

(i) Toute assertion RAINS possède une unique configuration, obtenue à partir des lettres de codes représentant les catégories des entités présentes dans cette assertion (voir section 5.2.3 et tableau 5.1)

(ii) Au terme de l'étape décrite à la section 5.2.3, nous avons restreint les possibles configurations pour l'assertion RAINS que nous cherchons à obtenir. Pour cela nous nous sommes servi des entités hors-ontologie de domaine, qui figurent dans l'assertion-LN. Cette liste des configurations RAINS possibles joue un rôle capital dans FORSA et est désignée par les termes "***liste restreinte des configurations***" ou "***liste restreinte des configurations candidates***". Dans les algorithmes que nous proposons, elle est représentée par la variable $\mathcal{L}$.

(iii) Pour chaque assertion RAINS, il existe des conditions de validité, prescrites par l'ontologie de domaine (voir section 4.5 relative à la sémantique de RAINS).

A. Conservation ou non des *undetermined properties*

Pour cette étape de résolution des assertions RAINS, la configuration des assertions est une donnée essentielle. Dans la liste des assertions obtenues à l'étape précédente, certaines sont susceptibles de contenir des *undetermined properties* (lettre de code **X**). Il existe des cas pour lesquels nous considérons que ces propriétés ont une sémantique et d'autres non. Avant d'exploiter les assertions candidates, nous les passons par ce filtre qui fonctionne de la manière énoncée ci-après. Dans le cas où (par ***ordre de priorité***) :

(a) L'assertion dispose déjà d'au moins une propriété (*datatype*, *object* ou *boolean property*) alors toutes les propriétés indéterminées sont supprimées

(b) L'assertion contient une seule propriété indéterminée, cette dernière est conservée. La lettre de code **X**, dans la configuration, est ainsi remplacée par **O**.

(c) L'assertion dispose de plusieurs propriétés indéterminées, alors on regarde les X comme des *object properties* (**O**) et on conserve celle(s) qui permet(tent) d'avoir une configuration RAINS valide. Dans le cas où on n'arrive pas à obtenir des configurations RAINS valides, on conserve la configuration initiale en l'état.

Appliquées aux assertions RAINS candidates de notre exemple, on obtient :

1. Assertion candidate 1 : **DCNXKFX** $\rightarrow$ **DCNKF** (car on a une *datatype property* symbolsée par **D**). L'assertion RAINS proprement dite devient
(ρ_0) `"rated speed" "Lifting platform" "not" "greater than" "0.15"`
2. Assertion candidate 2 : **DINXKFX** $\rightarrow$ **DINKF** (idem), d'où
(ρ_1) `"rated speed" "LIFTING_GEAR" "not" "greater than" "0.15"`

B. Résolution de l'assertion RAINS proprement dite

Ainsi, chacune des assertions RAINS obtenues à l'étape précédente rentre nécessairement dans une et une seule des catégories ci-après :

1. Sa configuration ***figure dans la liste*** restreinte des configurations ($\mathcal{L}$) et cette assertion est ***sémantiquement valide***. **Notation de la catégorie :** LV (liste et valide)
2. Sa configuration ***figure dans la*** $\mathcal{L}$ et cette assertion ***n'est pas sémantiquement valide***. **Notation de la catégorie :** $L\overline{V}$ (liste et non valide)
3. Sa configuration ***ne figure pas dans*** $\mathcal{L}$ et elle est, par conséquent[7], non sémantiquement valide. **Notation de la catégorie :** $\overline{L}\overline{V}$ (non liste et non valide)

Rendu à ce niveau, nous supposons que les assertions RAINS candidates ont été partitionnées suivant les trois catégories LV, $L\overline{V}$ et $\overline{L}\overline{V}$ décrites ci-dessus. Pour déterminer l'assertion RAINS finale en utilisant les assertions présentes dans ces trois catégories, nous avons recours à l'algorithme B.1 (annexe B, page 190).
Dans cet algorithme (qui à son tour fait appel aux algorithmes B.2 et B.3 de l'annexe B) :

— Pour déterminer l'assertion RAINS finale, on privilégie l'utilisation des éléments de la catégorie des assertions sémantiquement valides et dont les configurations sont dans $\mathcal{L}$ (catégorie LV). Si et seulement si cette catégorie est vide, alors on fait appel aux éléments de la catégorie $L\overline{V}$. De même, si et seulement si $L\overline{V}$ est vide, on utilise la catégorie $\overline{L}\overline{V}$.
— Que l'on se serve de la catégorie LV ou de $L\overline{V}$ ou de $\overline{L}\overline{V}$, le processus de résolution obéit à un même schéma : (i) on identifie le sous-ensemble $\mathcal{L}'$ de $\mathcal{L}$, qui contient

7. Rappelons que la liste des configurations restreintes ($\mathcal{L}$) comprend ***toutes*** les configurations RAINS qui contiennent le motif décrit par les entités non ontologiques de l'assertion. Ainsi rendu à cette étape, si la configuration d'une assertion RAINS candidate est absente de la liste restreinte de configurations, alors elle l'est aussi de la liste complète des configurations RAINS. De ce fait, sa sémantique n'est pas définie.

les configurations RAINS majoritairement plébiscitées, après un ***vote***, par les assertions de la catégorie (ii) L'assertion finale est déterminée en se servant des éléments de $\mathcal{L}'$.

Dans la suite, en fonction que l'on se situe dans la catégorie LV, $L\overline{V}$ ou $\overline{L}\overline{V}$, nous indiquons le processus de la détermination de l'assertion RAINS finale. Notamment nous présentons la méthode d'obtention de $\mathcal{L}'$ et la manière dont $\mathcal{L}'$ nous permet d'obtenir l'assertion RAINS finale. Cela est fait à la lumière des algorithmes B.1 - B.3 aux pages 190 à 192, dans l'annexe B.

Catégorie LV.

1. ***Obtention de $\mathcal{L}'$.*** Rappelons que $\mathcal{L}'$ est un sous-ensemble de l'ensemble restreint des configurations candidates (et noté $\mathcal{L}$). Un élément de $\mathcal{L}$ est dans $\mathcal{L}'$ si et seulement s'il est majoritairement ***plébiscité*** par les configurations des assertions RAINS candidates (dans ce cas de la catégorie LV). Ce vote majoritaire est réalisé à travers la ligne ❶ de l'algorithme B.1.
2. ***Détermination de l'assertion RAINS finale.*** Une assertion RAINS candidate est dans LV si elle a une configuration RAINS autorisée et si elle est sémantiquement valide. Ainsi, lorsque nous avons à faire à la catégorie LV nous utilisons une méthode de ***sélection*** de la "meilleure" assertion. Cette sélection est réalisée à travers la boucle ❷ et la ligne ❸ de l'algorithme B.1. À cette ligne ❸ on peut voir qu'on se sert d'un algorithme de tri pour classer les candidates. Dans cette catégorie, ce tri utilise des scores qui sont calculés, pour chaque assertion candidate, en prenant la moyenne des scores de *string similarity* entre les entités (dans les syntagmes) de l'assertion-LN et les entités ontologiques de cette assertion RAINS candidate.

Catégorie $L\overline{V}$.

1. ***Obtention de $\mathcal{L}'$.*** Dans cette catégorie $\mathcal{L}'$ s'obtient d'une manière identique (ligne ❹ de l'algorithme B.1) à celle utilisée pour obtenir $\mathcal{L}'$ dans la catégorie LV. Ceci vient du fait que les configurations des assertions de $L\overline{V}$ sont par définition présentes dans $\mathcal{L}$.
2. ***Détermination de l'assertion RAINS "provisoire".*** Dans cette catégorie nous parlons d'assertion RAINS provisoire parce que les assertions RAINS candidates de cette catégorie sont par définition non valides. Par conséquent, il y aura des opérations supplémentaires pour obtenir l'assertion RAINS finale. Cette assertion RAINS provisoire est obtenue au terme d'un processus de sélection décrit par la boucle ❺ et la ligne ❻ de l'algorithme B.1. De manière similaire au processus de sélection dans la catégorie LV, ici aussi on utilise un *ranking* des assertions candidates. Cependant, il existe une différence dans la manière dont les scores des candidates sont calculés. Cela est dû au fait que dans cette catégorie les candidates sont non valides. ***Heuristiquement***, nous imputons cette invalidité aux propriétés contenues

dans les assertions candidates et utilisons uniquement les *matching scores* des entités ayant permis d'obtenir les concepts ou les individus. Cette heuristique vient du fait que les prédicats en LN utilisent parfois des termes très différents des prédicats formels. Nous avons par exemple vu qu'avec l'ontologie IfcOWL (voir pages 38 et suivantes et section 4.4), un prédicat en LN pouvait donner lieu, après formalisation, à une chaîne de propriétés ***dont les libellés sont parfois différents des mots employés dans les normes en LN***. Ainsi l'identification des prédicats formels n'est pas nécessairement immédiate ; l'immédiateté signifiant qu'un terme en LN corresponde à un terme formel. Dans un tel contexte (c'est-à-dire non validité des assertions candidates), le ***matching score*** entre les termes de l'assertion NL et les prédicats de l'ontologie peut être non significatif. Or, les conditions de validité sont principalement supportées par les prédicats (*object* et *dataype properties*) présents dans les assertions.

Catégorie $\overline{L}\overline{V}$.

1. ***Obtention de $\mathcal{L}'$.*** Par définition, les assertions candidates de cette catégorie ont une configuration non présente dans $\mathcal{L}$. La configuration d'une assertion RAINS candidate ne peut donc pas directement plébisciter une configuration de $\mathcal{L}$. Chaque configuration candidate est donc amenée à voter pour les configurations de $\mathcal{L}$ les plus proches d'elle. Cette proximité est calculée en utilisant la distance de d'édition aussi connue sous le nom de ***distance de Levenshtein***[8] (voir lignes ❽ et ❾ de l'algorithme B.1).
2. ***Détermination de l'assertion RAINS "provisoire".*** Pour des raisons identiques à celles mentionnées lors du traitement de la catégorie $L\overline{V}$, l'assertion RAINS obtenue en exploitant les candidates de cette catégorie est provisoire. Toutefois, cette catégorie présente une différence fondamentale avec les deux autres catégories. Ici, il est nécessaire de trouver la configuration RAINS (variable dénommée `rainsConfig` dans les algorithmes B.2 - ligne ⑤ - et B.3 - en paramètre d'entrée) qui sied le mieux aux assertions de cette catégorie. `rainsConfig` servira de moule pour façonner l'assertion optimale (dénommée `candidateAss` dans les algorithmes B.2 - lignes ③ et ④ - et B.3 - en paramètre d'entrée) de $\overline{L}\overline{V}$. Pour cela : (*i*) À partir de $\mathcal{L}'$, on obtient ***l'une*** des configurations (variable `candidateConfig` dans l'algorithme B.2 - ligne ② - et dans l'algorithme B.3 - paramètre d'entrée) des assertions de $\overline{L}\overline{V}$ qui se rapprochent le plus des configurations de $\mathcal{L}'$ (*ii*) Ensuite, on détermine l'assertion `candidateAss` de $\overline{L}\overline{V}$, ayant pour configuration `candidateConfig` et ayant le meilleur *matching score* (voir ligne ③ de l'algorithme B.2) (*iii*) Enfin, à l'aide de la liste des modifications (variable `editSequence` dans l'algorithme B.3 - ligne ①) pour transformer `candidateConfig` en `rainsConfig`, on transforme `candidateAss` en une assertion RAINS ayant pour configuration `rainsConfig`. Cette transformation est

8. https ://fr.wikipedia.org/wiki/Distance_de_Levenshtein

décrite par la boucle ② de l'algorithme B.3.

L'algorithme B.1, qui nous permet de déterminer l'assertion RAINS finale ou provisoire (variable `assertionRains`) d'une assertion-LN, peut déboucher sur la valeur *null*. Cela ne peut se produire que lorsqu'on travaille avec les assertions de la catégorie $\overline{L}\overline{V}$ (voir ligne ⑥ de l'algorithme B.2). ***Une assertion-LN ayant débouché sur une assertion RAINS** null **sera écartée de la suite du processus de formalisation.***

Les annotations sont un cas à part d'assertions RAINS. Nous décrivons ci-dessous la résolution des annotations.

Cas des annotations. Les annotations RAINS occupent une place particulière en matière de configurations. Dans la formulation des exigences en langage naturel, il n'est pas indiqué que tel groupe de mots joue le rôle d'une annotation. Nous nous appuyons donc sur des heuristiques pour déterminer des annotations. Tout d'abord nous recherchons les potentielles annotations uniquement dans les assertions RAINS étant constituées d'une seule entité. Cette unique entité n'a d'intérêt pour l'approche FORSA que si elle désigne un ***individu***. Cela vient du fait que les valeurs possibles des annotations sont soient des individus, soient des litéraux[9] comme l'illustre la figure 4.2 en page 92. Un individu est une valeur possible d'annotation au regard de l'ontologie de domaine si la requête SPARQL suivante fournie un résultat non-vide.

```
SELECT ?annotationName
WHERE { ?annotationName rdfs:domain ?annotation.
        ?annotation rdfs:subClassOf* :Annotation.
        ?annotationName rdfs:range ?enum.
        :annotationValue rdf:type ?type.
        ?type rdfs:subClassOf* ?enum. }
LIMIT 1
```

Dans cette requête, `:annotationValue` est l'URI de l'individu en question. Cette requête recherche les propriétés dont le *domain* (au sens `rdfs:domain`) est une sous classe de la classe `:Annotation` et dont le *range* (au sens `rdfs:range`) est une énumération dont la liste des items contient `:annotationValue` (c'est-à-dire l'individu). Si cette requête retourne une propriété, alors nous considérons que cette propriété est le nom de l'annotation. Ainsi, la version RAINS de l'assertion-LN est l'annotation **`@ AnnotationName (AnnotationValue)`**.

Si nous exécutons cette requête, envers l'ontologie de la figure 4.2, avec l'individu :

- `:INDIVIDUAL_HOUSE`, on obtient la propriété `:buildingUse` et donc l'annotation RAINS : **`@ Building use (Individual house)`**
- `:FIRE_SAFETY`, on obtient la valeur `:domainType` et par conséquent l'annotation RAINS : **`@ Domain type (Fire safety)`**

9. Actuellement, FORSA n'intègre pas de mécanisme de détection automatique des annotations dont les valeurs sont des litéraux.

Pour une assertion composée d'une seule entité qui ***n'est pas un individu*** ou alors qui est un ***individu ne pouvant être la valeur d'une annotation***, alors cette assertion est considérée comme étant *null*.

Au terme de cette étape, nous obtenons une assertion RAINS qui est :

— Soit une annotation RAINS
— Soit une assertion nulle, et dans ce cas nous l'écartons du processus de formalisation de l'exigence
— Soit valide (obtenue en utilisant la catégorie LV)
— Soit non valide (obtenue en ayant recours à la catégorie $L\overline{V}$ ou à la catégorie $\overline{L}\overline{V}$). Dans ce cas, nous avons vu que nous ne conservons que les concepts et/ou individus de l'assertion, en plus des éventuelles entités non ontologiques (comparateurs, marques de négation, etc.). Si la configuration de l'assertion fait mention de propriétés pour lier les concepts/individus, ou pour lier les concepts aux littéraux, il nous revient donc de trouver des propriétés appropriées dans l'ontologie de domaine, pour établir ces liaisons. Nous apportons la réponse à cette question dans la sous-section suivante (page 132).

C. Identification des relations entre entités

Rendu à cette étape de la transformation d'une assertion-LN en une assertion RAINS, nous venons d'obtenir une première assertion RAINS au terme de l'algorithme B.1. Si cette assertion est valide, cette étape que nous décrivons n'est pas nécessaire. Nous n'avons recours à l'identification des relations entre les entités de l'assertion que si celle-ci n'est pas valide. Pour cette opération, nous considérons pour acquis les entités de l'assertion qui sont soient des entités non ontologiques de l'assertion (comparateurs, marques de négation, etc. - voir section 5.2.2), soient des concepts ou individus. Ainsi, il nous reste à trouver les propriétés, telles que suggérées par la configuration de l'assertion. En fonction de la position des propriétés dans la configuration, trois cas de figure peuvent se présenter. En effet la propriété peut relier (voir *listing* 4.1) :

1. Deux concepts. C'est le cas dans les *ordered* et les *inverted object assertions*
2. Un concept et un individu. On peut observer ce cas dans les *ordered* et les *inverted object assertions with individual*
3. Un concept et un littéral à travers une *datatype property*. Ce cas est illustré par les *data assertions*

Chacun de ces cas donne lieu à une question, dont la construction de la réponse sera adaptée à chaque cas, à savoir : "Par quelle(s) propriété(s) peut-on lier les entités A et B ?". De manière formelle, cela peut se traduire ainsi ***"Quelle requête SPARQL permet d'obtenir la liste des propriétés capables de relier les entités A et B ?"***. Pour nos trois cas, nous proposons trois fonctions, une pour chaque cas de figure, pour l'obtention de cette requête. Ces fonctions sont détaillées par les *listings* 5.2, 5.3 et 5.4.

Listing 5.2 – Code (écrit en java) pour l'écriture de la requête SPARQL permettant d'obtenir la liste des propriétés pouvant relier deux concepts

```
1 private static String getPathsQueryClassToClass(String source, String target,
      int pathLength) {
    String propertyPath = "";
3   String currentDomain = "?Dp1";
    String finalRange = "?Rp" + pathLength;
5   String currentRange;
    String properties = "";
7
    ①String query = "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        SELECT ?props WHERE{?subj rdfs:subClassOf* ?Dp1. ?obj rdfs:subClassOf* "
         + finalRange + ".?propertyPath}";
9   for (int i = 1; i <= pathLength; i++) {
        ②String varI = "?p" + i;
11      String varTempI = "?temp" + i;
        ③properties += varI + " ";
13      ④String domainTemp =" "+ varI +" rdfs:domain " + currentDomain +".";

15      if (i > 1) {
                ⑤domainTemp = "{{" + domainTemp + "} UNION {" + varTempI + "
                    rdfs:subClassOf* " + currentDomain + ". " + varI + " rdfs:
                    domain " + varTempI + "}}.";
17      }
        ⑦propertyPath += " " + domainTemp;
19      if (i == pathLength) {
      ⑧propertyPath += " " + varI + " rdfs:range " + finalRange + ".";
21      } else {
      currentRange = "?Dp" + (i + 1);
23    currentDomain = currentRange; //The range of ?pi is equal to the domain
          of ?p(i+1)
      ⑨propertyPath += " " + varI + " rdfs:range " + currentRange + ".";
25      }
    } // Fin boucle for
27  query = query.replace("?subj", source).replace("?obj", target);
    query = query.replace("?propertyPath", propertyPath).replace("?props",
        properties);
29  return query;
}
```

Le *listing* 5.2 est consacré à l'écriture de la requête SPARQL pour la liaison de deux concepts. La fonction qu'il présente possède trois paramètres en entrée : `source` qui est le concept en début de chemin, `target` qui est le concept que l'on veut relier à partir de `source` et `pathLength` qui est la longueur du chemin entre `source` et `target`. En pratique, cette fonction est appelée pour `pathLength` $= 1$. Si en exécutant la requête SPARQL résultante de cet appel de fonction, on ne trouve pas de chemin de longueur 1, alors on fait appel à cette fonction avec la valeur 2 pour `pathLength`. Ainsi de suite jusqu'à l'obtention d'au moins un chemin, entre `source` et `target`, à l'exécution de la requête. En

pratique, nous nous limitons à pathLength = 5. Si nous n'avons pas trouvé de chemin, l'assertion est écartée.

Les principales étapes de cette fonction sont les suivantes :

Ligne ① : On forme le corps de la requête, query dans lequel on spécifie qu'on veut sélectionner (clause SELECT) des propriétés (variable SPARQL ?props), sous les conditions spécifiées dans le WHERE. Ces conditions sont : (*i*) source, représentée par la variable SPARQL ?subj, est une sous-classe de ?Dp1 (*ii*) target, représentée par ?obj, est une sous-classe de ?Rpn, où *n* est la valeur de pathLength et (*iii*) Il doit avoir un chemin, représenté par ?propertyPath, entre ?Dp1 et ?Rpn

Lignes ⑥, ⑦ et ⑧ : On construit la valeur de la chaîne de caractères propertyPath, qui est le libellé de la variable SPARQL ?propertyPath, selon le principe : deux propriétés ?pi et ?pi+1 ne peuvent être consécutives, sur le chemin allant de source à target, que si ?pi+1 ∈ out-properties(?pi) (voir définition 7 en page 91). Ce chemin commence par une propriété ?p1 telle que ?Dp1 ∈ domain(?p1) et se termine par une propriété ?pn telle que ?Rpn ∈ range(?pn).

Ligne ③ : Le libellé de ?props, dans le corps de la requête SPARQL, est donné par la variable-java properties, elle-même formée par la liste des variables SPARQL de la forme ?p1, ?p2, etc. (ligne ②)

Pour avoir une meilleure idée du résultat de l'appel de cette méthode, nous reproduisons ci-dessous les résultats pour les valeurs 1 et 2 de pathLength :

— Pour pathLength = 1 on a :

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?p1  WHERE{<SOURCE> rdfs:subClassOf* ?Dp1. <TARGET> rdfs:
    subClassOf* ?Rp1.  ?p1 rdfs:domain ?Dp1. ?p1 rdfs:range ?Rp1.}
```

— Pour pathLength = 2 on a :

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?p1 ?p2  WHERE{<SOURCE> rdfs:subClassOf* ?Dp1. <TARGET>
    rdfs:subClassOf* ?Rp2.  ?p1 rdfs:domain ?Dp1. ?p1 rdfs:range ?
    Dp2. {{ ?p2 rdfs:domain ?Dp2.} UNION {?temp2 rdfs:subClassOf*
    ?Dp2. ?p2 rdfs:domain ?temp2}}. ?p2 rdfs:range ?Rp2.}
```

Listing 5.3 – Code (java) pour l'écriture de la requête SPARQL permettant d'obtenir la liste des propriétés pouvant relier un concept à un individu

```
private static String getPathsQueryClassToIndividual(String source, String
    target, int pathLength) {
        String propertyPath = "";
        String currentDomain = "?Dp1";
        String finalRange = "?Rp" + pathLength;
        String currentRange;
        String properties = "";
        ❶String query = "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
            PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> SELECT ?
```

```
            props WHERE{" + "?subj rdfs:subClassOf* ?Dp1. ?obj rdf:type ?t. ?t
            rdfs:subClassOf*  " + finalRange + ".?propertyPath}";
        for (int i = 1; i <= pathLength; i++) {
                    //Idem à la boucle du listing 5.2
        }
        //Idem aux instructions du listing 5.2
        return query; }
```

Le *listing* 5.3 consacré à l'écriture de la requête SPARQL pour la liaison de d'un concept à un individu, est très similaire au *listing* 5.2. La seule différence réside dans la constitution de la requête (ligne ① du *listing* 5.2 et ligne ❶ du *listing* 5.3). Sachant que `target` désigne un individu, la condition de subsomption "`target`, représentée par `?obj`, est sous classe de `?Rpn`, où *n* est la valeur de `pathLength`" est remplacée par ***"l'individu* `target`*, représenté par* `?obj`*, est d'un type* `?t`*, telle que* `?t` *est une sous classe de* `?Rpn`"**. Notons que si le chemin à rechercher va d'un individu à un concept, alors les rôles de `source` et `target` seraient inversés dans la fonction.

Listing 5.4 – Code (java) pour l'écriture de la requête SPARQL permettant d'obtenir la liste des propriétés pouvant relier un concept à une propriété

```
private static String getPathsQueryClassToProperty(String source, String target
    , int pathLength) {
        ①if (pathLength == 1) {
            return target;
        }
        String propertyPath = "";
        String currentDomain = "?Dp1";
        String finalRange = "?Dp" + pathLength;
        String currentRange;
        String properties = "";

        ② String query = "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
             SELECT ?props WHERE{" + "?subj rdfs:subClassOf* ?Dp1. ?prop
            rdfs:domain " + finalRange + " UNION " + "?temp rdfs:subClassOf* " + finalRange +
            ". ?prop rdfs:domain ?temp .?propertyPath}";
        for (int i = 2; i <= pathLength; i++) {
            String varI = "?p" + (i - 1);
            String varTempI = "?temp" + (i - 1);
            //Idem aux instructions du listing 5.2
            ③if (i == pathLength) {
                propertyPath +=" " + varI +" rdfs:range " + finalRange +".";
            } else {
                currentRange = "?Dp" + i;
                currentDomain = currentRange; //The range of ?p(i-1) is equal
                    to the domain of ?pi
                propertyPath +=" " +varI +" rdfs:range " +currentRange +".";
            }
        }
        //Idem aux instructions du listing 5.2
        return query;}
```

Entre la détermination de la requête SPARQL pour la recherche des chemins entre deux concepts (*listing* 5.2) et la requête pour la recherche des chemins entre un concept et une propriété, il y a très peu de différences. Notons que lorsque nous parlons de chemin entre un concept `source` et une propriété `target`, il s'agit de l'ensemble des propriétés à travers lesquelles on peut cheminer, dans le graphe dénoté par l'ontologie de domaine, de `source` jusqu'à atteindre une propriété `p` telle que `p` $\in$ `in-properties(target)` (voir définition 8 en page 91). La longueur du chemin, `pathLength`, tient compte de la propriété cible `target`. Autrement dit, si `pathLength` $= 1$, alors la seule propriété du chemin est `target` (ligne ①). Si le chemin a une longueur supérieure à 1 alors on reprend le même principe que lors de la détermination du chemin entre deux concepts (*listing* 5.2). Toutefois, dans ce cas, la $n^{ème}$ propriété du chemin, `?pn`, est égale à `target`.

Au terme de cette étape nous avons soit une assertion RAINS valide, soit une assertion RAINS vide (dans le cas où les différents élagages ont permis de conclure que cette assertion était sémantiquement vide). Une assertion vide ne sera pas exploitée pour la suite des traitements.

Revenons à notre exemple. Nos données sont :

— La liste des configurations possibles est : $\mathcal{L} = \{$**CDNKF**, **DCNKF** et **CODNKF**$\}$.
— Les assertions candidates

1. (ρ_0) `"rated speed" "Lifting platform" "not" "greater than" "0.15"`
   Dont la configuration est **DCNKF**

2. (ρ_1) `"rated speed" "LIFTING_GEAR" "not" "greater than" "0.15"`
   Dont la configuration est **DINKF**

On constate que (ρ_0) appartient à la catégorie LV. En effet, sa configuration est dans la liste restreinte et elle est valide. Au regard de la figure 4.1, on a `domain(:ratedSpeed)` $=$ $\{$`:LiftingPlatform`$\}$ et `in-properties(xsd:double)` $= \{$`:length`, `:panelwidth`, `:ratedSpeed`$\}$. Pour qu'une *datatype property* de configuration **DCNKF** soit valide, il faut que le concept dénoté par le symbole **C** appartienne à l'ensemble `domain(:datatypeProperty)` où l'entité `:datatypeProperty` est la propriété dénotée par la lettre de code **D**. De plus, il faut que `:datatypeProperty` appartienne à `in-properties(xsd:double)` [10]. On peut remarquer que c'est le cas ici. Par contre, (ρ_1) appartient à la catégorie $\overline{LV}$. En effet, sa configuration n'est pas dans la liste restreinte.

Par conséquent nous avons $LV = \{(\rho_0)\}$ et $\overline{LV} = \{(\rho_1)\}$. LV étant non vide, c'est de la que doit provenir l'assertion RAINS. LV étant un singleton, aucun *ranking* n'est nécessaire. L'assertion RAINS finale est donc (ρ_0).

10. La classe du littéral dénoté par **F** est `xsd:double`, étant donné que F symbolise un flottant.

5.3 Résolution de l'antécédent de la règle RAINS

Comme son nom l'indique, le but de cette étape est de construire l'antécédent de notre règle RAINS. Les éléments qui nous seront utiles pour cette tâche sont :

(i) Les assertions RAINS obtenues des assertions en LN de l'exigence

(ii) Le ***prédicat principal*** de l'exigence.

Le but de l'approche FORSA est de réécrire une exigence réglementaire sous la forme d'une règle RAINS. Cependant, cette règle doit servir à pointer du doigt un ***défaut*** sur un ***objet précis***. Or, affirmer qu'un objet présente un défaut, c'est dire que cet objet viole une contrainte. Il est donc nécessaire de détecter le terme de l'exigence, à manipuler, pour pouvoir reformuler la règle de sorte que l'antécédent n'ait la valeur de vérité "vraie" qu'en cas de violation de l'exigence. Ce terme à manipuler constitue ce que nous appelons ***prédicat principal***. Quant au concept de la règle RAINS qui va porter sceau de non-conformité, nous le qualifions de ***concept principal***. La détection du concept principal est présenter à la section 5.4.1.
Comme le montre la figure 5.1, présentant les principales étapes de FORSA, cette étape prend en entrée l'exigence réglementaire orignale, en plus des assertions RAINS obtenues de celle-ci. L'exigence réglementaire nous permettra d'obtenir les dépendances syntaxiques entre les mots de l'exigence qui la composent. Ces dépendances sont les éléments de base pour la détermination du prédicat principal.

5.3.1 Identification du prédicat principal

Au regard des dépendances syntaxiques, nous considérons que le prédicat principal est le **verbe** ***racine*** ou le ***copule*** (parfois appelé ***verbe d'état***) rattaché à la ***racine*** de la phrase réglementaire. Dans l'univers des dépendances syntaxiques, la racine (en anglais *root*) est la relation grammaticale qui permet de déterminer le point d'ancrage de la phrase [De Marneffe & Manning 2008]. Ce point d'ancrage étant lui aussi appelé racine. Des outils de TALN comme le Stanford parser fournissent, parmi la liste des dépendances syntaxiques, une dépendance sous la forme ***root(ROOT-***0***, racinePhrase-***n***)*** qui nous indique que :

— Nous sommes en présence de la dépendance racine dont le libellé est *root*

— Cette dépendance lie un mot fictif ***ROOT***, qui est le zéroième terme de la phrase (donc ROOT n'est pas un mot de la phrase), au mot ***racinePhrase***, qui est le $n^{ème}$ mot de la phrase.

Dans la dépendance `root(ROOT-0, racinePhrase-`n`)`, `racinePhrase` peut être un verbe [11] ou un complément d'un verbe d'état. La relation entre ce complément et le verbe d'état est fournie par la dépendance syntaxique ***cop(racinePhrase-***n***, copula-***m***)*** où `copula` est le verbe d'état et m sa position dans la phrase.

11. Comme nous l'avons déjà mentionné, on reconnait un verbe à son étiquette morpho-syntaxique : VB, VBD, VBG, VBN, VBP ou VBZ.

En prenant notre exemple ***"The rated speed of the lifting platform shall not be greater than 0,15 m/s"*** nous obtenons les dépendances syntaxiques qui sont telles que présentées par la figure 5.3. Compte tenu de la façon dont nous identifions le prédicat principal, nous pouvons dire que le verbe ***be***, est le prédicat principal.

Étiquettes morpho-syntaxiques

The/**DT** rated/**VBN** speed/**NN** of/**IN** the/**DT** lifting/**NN** platform/**NN** shall/**MD** not/**RB** be greater/**JJR** than/**IN** 0,15/**CD** m/s/**NN**

Dépendances syntaxiques

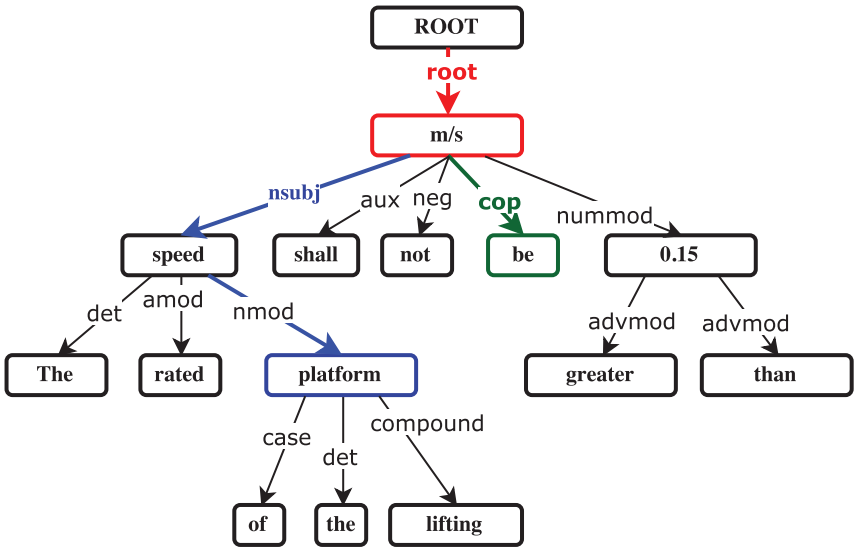


FIGURE 5.3 – Étiquettes morpho-syntaxiques et dépendances syntaxiques de la norme-exemple, obtenues avec le Stanford parser

Lors de la présentation de la notion de prédicat principal, nous avons souligné le fait que cet élément nous permettra de donner à la future règle RAINS, une forme lui permettant d'être une règle de détection de non-conformité. Nous décrivons l'utilisation du prédicat principal pour la réalisation de cette tâche dans la section ci-après.

5.3.2 Mise en évidence de la non-conformité

Nous présentons tout d'abord l'idée qui sous-tend la mise en évidence de la non-conformité. Supposons l'obligation $\mathcal{O}_1$ *"You must have an X"* (en français "Vous devez posséder un (objet) X"). S'il était demandé de fournir une formulation équivalente de cette phrase de manière à expliciter les conditions de violation de l'obligation, on ***pourrait*** obtenir la reformulation *"If you* **don't** *have an X then you are not compliant with law"* (en français "Si vous n'êtes pas en possession d'un (objet) X alors vous êtes en infraction."). Si l'on reprend l'exercice avec l'interdiction $\mathcal{I}_1$ *"You must* **not** *own a Y"* (en français

"Vous ne devez pas être en possession d'un (objet) Y"), on pourrait obtenir la règle sous forme d'une violation *"If you own a Y then you are not compliant with law"* ("Si vous êtes en possession d'un (objet) Y alors vous êtes en infraction"). Que ce soit dans le cas de l'obligation $\mathcal{O}_1$ ou de l'interdiction $\mathcal{I}_1$, la mise en évidence de la violation s'appuie sur un mécanisme similaire. En effet, pour expliciter la violation :

— Dans une obligation, il est possible d'**ajouter *une marque de négation*** au prédicat principal (reconnaissable dans l'exemple par un soulignement) de cette obligation

— Dans une interdiction, il est possible d'**ôter *une marque de négation*** au prédicat principal de cette interdiction

Toutefois, lorsque la règle comporte un marqueur d'interdiction (par exemple *"It is forbidden"*, *"It is prohibited"*, *"are forbidden"*, etc.), cette dernière joue le rôle d'une négation. Autrement dit, il suffit de retirer le marqueur d'interdiction pour reformuler la phrase sous une forme mettant en exergue la violation. Pour illustration, on peut reformuler $\mathcal{I}_2$ *"It is* **forbidden** *to own a Y"* (en français "Il est interdit d'être en possession d'un (objet) Y") en *"If you own a Y then you are not compliant with law"* - c'est-à-dire comme $\mathcal{I}_1$.

On peut faire le même constat sur des phrases plus complexes. Pour illustration la phrase $\mathcal{O}_2$ *"If you own an X and that you know how to use a Y, then you must have a Z."* peut être reformulée en *"If you own an X and that you know how to use a Y and you* **don't** *have a Z then you are not compliant with law."*. Une fois de plus, on constate que la reformulation de $\mathcal{O}_2$ s'appuie essentiellement sur la négation du prédicat principal.

Algorithme 5.1 Négation du prédicat principal dans les assertions RAINS

Données : `mainPredicate, naturalAssertions, naturalAssToRAINSAssMap, rainsConfigToNegatedRAINSConfigMap`

Résultat : `rainsAssertions`

```
rainsAssertions = ∅;
rainsAss = naturalAssToRAINSAssMap.get(nlAssertion);
pour nlAssertion ∈ naturalAssertions faire
    ① si nlAssertion.contains(mainpredicate) alors
        config = getConfiguration(rainsAss);
        negatedConfig = rainsConfigToNegatedRAINSConfigMap.get(config);
        ② negatedRAINSAss = addOrRemoveNegMark(rainsAss, negatedConfig);
        ③ rainsAssertions.add(negatedRAINSAss);
    sinon
        ④ rainsAssertions.add(rainsAss);
Retourner rainsAssertions;
```

Application dans FORSA

Dans cette section nous décrivons comment l'idée de l'obtention d'une règle de non-conformité à partir de la règle originale, présentée dans la section précédente, a été mise

en oeuvre dans FORSA. Cette mise en oeuvre est effectuée au travers de l'algorithme 5.1 ci-dessus.

Dans cet algorithme, le point d'ancrage est la ligne ①. Elle met en évidence le fait que toute assertion-LN (variable `nlAssertion`) extraite de l'exigence réglementaire rentre nécessairement dans l'une des catégories suivantes : (*i*) soit elle contient le prédicat principal, (*ii*) soit elle ne fait pas mention de ce dernier. Dans le cas où `nlAssertion` fait usage du prédicat principal, alors il faut obtenir la négation de l'assertion RAINS (variable `rainsAss`) obtenue à partir de `nlAssertion`. Cette opération de négation est réalisée sur la base de la configuration (variable `config`) de l'assertion `rainsAss`. La négation, telle qu'opérée ici, s'appuie sur le fait que, pour chaque configuration `config`, et donc pour chaque type d'assertion RAINS, il existe une configuration, dénotée par la variable `negatedConfig` qui en est la négation. Cela peut se remarquer en regardant la syntaxe formelle de RAINS (*listing* 4.1). On peut voir que dans toutes les assertions RAINS comportant au moins une propriété, il y a de la place pour une marque de négation, cette dernière étant optionnelle (symbole `Neg?`). Si on prend l'exemple de la définition de `BooleanAssertion`, on voit qu'elle propose six configurations possibles : `CB`, `CNB`, `COB`, `CNOB`, `BC` et `BNC` (pour rappel : `C` = concept, `O`= *object property* ou *chain of object properties*, `B`= *boolean datatype property* et `N` = marque de négation). À l'aide de ces configurations, on peut construire les couples ***(clé, valeur)*** présentés dans le tableau 5.2 ; ce qui permet d'obtenir en quelque sorte une table de hachage. Dans cette table de hachage, on peut remarquer qu'à partir d'une configuration sans marque de négation, on obtient nécessairement une négation de configuration qui elle, comporte une marque de négation et vice versa. Ainsi, en fonction du fait que la négation de la configuration demande d'***ajouter*** ou d'***ôter*** une marque de négation, on ajoutera ou enlèvera une marque de négation de l'assertion RAINS (ligne ②). On obtient de ce fait la négation (`negatedRainsAss`) de l'assertion RAINS de départ (`rainsAss`). `negatedRAINSAss` fera donc partie du résultat final (ligne ③).

Tableau 5.2 – Exemple de couples (clé, valeur) entre les configurations RAINS et leur négation

Configurations	**Négation**
CB	CNB
CNB	CB
COB	CNOB
CNOB	COB
BC	BNC
BNC	BC

Dans le cas où l'assertion-LN ne porte pas de marque de négation, sa version RAINS, `rainsAss`, est ajoutée sans modification à la liste des assertions RAINS en sortie (ligne ④). Comme nous l'avons formulé lors de la présentation de l'idée de la négation du

prédicat principal, lorsque l'exigence contient un marqueur d'interdiction, l'opération de négation du prédicat principal n'est pas effectuée.
Cependant, il peut arriver que ***le prédicat principal ne se retrouve dans aucune des assertions-LN***. Ceci est dû à l'élagage de certaines assertions-LN lors des étapes précédentes. Dans ce cas, ***la transformation de l'exigence en règle RAINS est interrompue et le service implémentant FORSA retourne un message d'erreur***.

Appliquons l'algorithme 5.1 à notre exigence-exemple *"The rated speed of the lifting platform shall not be greater than 0,15 m/s"*. Rappelons que le prédicat principal est le verbe ***be***. Aussi, de cette exigence, nous avons obtenu une assertion-LN
(τ_0)⟨`The rated speed of the lifting platform`, `shall not be`, `greater than 0,15 m/s`⟩.
À partir de (τ_0) nous avons obtenu l'assertion RAINS
(ρ_0) `"rated speed" "Lifting platform" "not" "greater than" "0.15"`.
En suivant les étapes de l'algorithme 5.1, on a :

(i) (τ_0) contient le prédicat principal

(ii) `DCNKF` est le configuration de (ρ_0)

(iii) La négation de cette configuration est `DCKF`

(iv) La négation de (ρ_0) s'obtient en enlevant la marque de négation. Le résultat de cette opération est
($\bar{\rho}_0$) `"rated speed" "Lifting platform" "greater than" "0.15"`

($\bar{\rho}_0$) est donc le seul élément de la liste des assertions RAINS de notre exigence-exemple.

5.3.3 Combinaison des assertions RAINS

Le but de cette étape est de regrouper les assertions RAINS pour constituer l'antécédent de la règle RAINS. Les traitements effectués à cette étape sont exposés par l'algorithme 5.2 (page 142).
Dans cet algorithme, on remarque que chacune des assertions de la liste des assertions RAINS se trouve dans l'un des trois cas suivants :

1. Soit c'est une annotation RAINS[12] (ligne ❶ de l'algorithme 5.2). Dans ce cas, on la place en début de l'antécédent comme l'exigence la syntaxe de RAINS (voir la ligne 1 du *listing* 4.1, présentant la syntaxe de RAINS).
2. Soit c'est une assertion proprement dite (c'est-à-dire *double concept*, *boolean*, *datatype* ou *object assertion*). Dans ce cas, le principal traitement à effectuer est celui de la résolution de co-référence. Cette opération intervient dans le cas où l'assertion utilise un concept précédemment mentionné dans une autre assertion (bloc ❷).

En outre, il est important de noter que toutes les assertions ne rentre pas dans la constitution de l'antécédent de la règle. On peut voir que l'algorithme dispose d'un mécanisme

12. Pour rappel, les assertions à une seule entité sont transformées en annotations. Voir page 131 et suivantes

de filtre incarné par la variable prune dont la valeur est exploitée à la ligne ❸. Exception faite des annotations et de la première assertion qui font systématiquement partie de l'assertion, les autres assertions doivent nécessairement contenir des concepts faisant références aux concepts déjà utilisés dans l'antécédent. En effet, ces co-références assurent la cohésion des assertions de l'antécédent. Si d'une ligne à l'autre il n'y a pas de co-référence entre concepts, cela veut dire que les assertions sont ***indépendantes***. L'algorithme 5.2 évite que l'on se retrouve avec des assertions "non-connectées".

Algorithme 5.2 Combinaison des assertions RAINS

```
Données : rainsAssertions
Résultat : antecedent
conceptToPositionMap = emptyMap(); count = 0;
first = true; SI oui ou non, c'est la première assertion de l'antécédent
antecedent = " ";
pour rainsAss ∈ rainsAssertions faire
    prune = true; Si oui ou non si on écarte une assertion
    temp = rainsAss;
    listEntities = getEntities(rainsAss);
    ❶ si listEntities.size() == 1 alors
        annotation = listEntities.get(0);
        antecedent = annotation + "\n" + antecedent; "\n" = retour chariot
    sinon
        pour entity ∈ listEntities faire
            count = count +1;
            si isAConcept(entity) alors
                ❷ si conceptToPositionMap.containsKey(entity) alors
                    coref = conceptToPositionMap.get(entity);
                    temp = temp.replace(entity, entity + coref);
                    prune = false;
                sinon
                    conceptToPositionMap.put(entity, count);
        si first alors
            introducer = random("If", "When", "Given");
            first = false;
            antecedent = antecedent + introducer + temp + "\n";
        sinon
            ❸ si !prune alors
                introducer = "And";
                antecedent = antecedent + introducer + temp + "\n";
Retourner antecedent;
```

Notre exigence-exemple comporte une assertion et aucune annotation. L'application de

l'algorithme de combinaison des assertions dans ce cas est immédiate. Elle produit en résultat l'antécédent :

```
If "rated speed" "Lifting platform" "greater than" "0.15"
```

Maintenant que nous disposons de l'antécédent de notre règle RAINS, nous pouvons conclure l'approche FORSA en présentant les mécanismes pour l'obtention du conséquent de la règle.

5.4 Résolution du conséquent de la règle RAINS

Tout comme la détermination de l'antécédent repose sur le prédicat principal, le conséquent est construit autour d'un élément qui est le ***concept principal***.

5.4.1 Identification du concept principal

Comme nous l'avons mentionné précédemment, le concept principal est le concept de la règle RAINS qui sera référencé par l'entité `"NonCompliant"` du conséquent. Contrairement au prédicat principal ; qui est un mot de l'exigence original, le concept principal est une entité RAINS. Il fait partie des entités RAINS présentes dans l'antécédent obtenu au terme de l'étape précédente.
Comme l'identification du prédicat principal (voir section 5.3.1), cette tâche s'appuie sur les dépendances syntaxiques. La terminologie utilisée ici est reprise du manuel des dépendances du Stanford parser [De Marneffe & Manning 2008]. Pour obtenir le concept principal :

1. Nous partons du noeud racine. Le noeud racine est le dépendant de la relation syntaxique `root(ROOT, racine)`.
2. Nous obtenons le ***sujet*** (variable `subject`) de la phrase. Le sujet correspond au dépendant de la relation syntaxique `nsubj(racine, subject)` ou `nbsubjpass(racine, subject)`. Le parser produit l'une ou l'autre dépendance selon que la phrase est à la ***voix active*** ou ***passive***.
3. À partir du `subject`, on effectue un ***parcours en pré-ordre*** de l'arbre des dépendances syntaxiques. Tout au long de ce parcours, on ne s'intéresse qu'aux ***noms*** [13]. Cela vient du fait que, lors de l'identification des entités ontologiques pour la formation des assertions RAINS, nous avons recherché les concepts uniquement à partir des syntagmes nominaux (voir pages 124 et suivantes). Pour chacun des noms rencontrés sur ce parcours préfixé, nous regardons dans la formalisation des syntagmes (décrite à la section 5.2.4) si ce nom à servi à l'obtention d'un concept [14]. Le premier concept que l'on obtient dans cet itération est le concept principal.

13. On reconnaît un nom à son étiquette morpho-syntaxique : NN, NNS, NNP ou NNPS.
14. Pour rappel certains noms peuvent être formalisés par des propriétés.

En appliquant cet algorithme à l'arbre des dépendances syntaxiques de la figure 5.3, à partir de la racine `m/s`, nous obtenons le sujet `speed`. Lorsque nous commençons notre parcours en pré-ordre à partir de ce noeud, nous obtenons les noms suivants :

— `speed` : ce nom a permis l'obtention de la *datatype property* `"rated speed"`. Nous continuons donc notre recherche

— `platform` : ce nom, quant à lui, rentre en compte dans l'obtention du concept `"Lifting platform"`. Ce dernier constitue donc notre concept principal.

5.4.2 Détermination de la référence du concept de non-conformité

Nous disposons déjà de l'antécédent de notre règle RAINS (voir section 5.3). Pour avoir la règle RAINS complète, il nous faut déterminer le conséquent de la règle. C'est le but de cette dernière phase de l'approche FORSA.

Algorithme 5.3 Détermination de la référence du concept de non-conformité envers le concept principal

```
Données : antecedent, mainConcept
Résultat : coreference
coreference = 0;
rainsAss = naturalAssToRAINSAssMap.get(nlAssertion);
pour assertion ∈ antecedent.split("\n") faire
    ① si not isAnAnnotation(assertion) alors
        pour entity ∈ getEntities(assertion) faire
            coreference = coreference +1;
            si entity.equals(mainConcept) alors
                Retourner coreference;
Retourner coreference;
```

Tout au long de la présentation de RAINS et de FORSA, nous avons insisté sur le fait que nous souhaitons reformuler les exigences sous forme de règles dédiées au contrôle de la conformité. Pour cela, le conséquent des règles formelles que nous écrivons permet de conclure que le concept principal de la règle est non-conforme. Ainsi, l'entité maitresse de l'assertion RAINS qui compose le conséquent de nos règles est de la forme

$$\texttt{"Non-compliant"}n$$

où `"Non-compliant"` est la classe OWL des objets présentant des défauts (voir figure 4.1) et n est l'entier permettant de référencer le concept principal de la règle RAINS. La valeur de n est déterminée par le rang du concept principal dans la liste des assertions RAINS. Cette opération est détaillée dans l'algorithme 5.3 ci-dessus. Maintenant que

nous disposons de la référence du concept "Non-compliant", le conséquent proprement dit s'obtient de la manière suivante :

Then it is "Non-compliant"n

La syntaxe formelle de RAINS, oblige à ouvrir le conséquent d'une règle par le mot clé Then. À ce mot on peut accoler une *single concept assertion* composée du concept de non-conformité suivi de la référence au concept principal.
Exécutons ces instructions sur notre exemple. L'antécédent de notre règle est
If "rated speed" "Lifting platform" "greater than" "0.15"
et le concept principal est l'entité "Lifting platform". L'algorithme 5.3 de détermination de la position de "Lifting platform" dans l'antécédent nous retourne la valeur 2. Le conséquent de la règle est alors : **Then "Non-compliant"2**

La règle RAINS obtenue par FORSA à partir de l'exigence *"The rated speed of the lifting platform shall not be greater than 0,15 m/s"* est :
If "rated speed" "Lifting platform" "greater than" "0.15"
Then "Non-compliant"2

5.5 Exemples d'application de FORSA

Dans cette section, nous illustrons le fonctionnement de l'approche FORSA sur quelques exigences réglementaires. Nous reprenons les normes mentionnées au chapitre 4 pour présenter le langage contrôlé RAINS.

Exemple 1 : *"The rated speed of the lifting platform shall not be greater than 0,15 m/s"*

Nous avons déroulé cet exemple tout au long de la présentation de FORSA. Dans cette section, nous regroupons les différents morceaux des traitements effectués.

1. Identification des assertions

Un seul triplet est extrait de cette exigence :
$(\tau_0)\langle$The rated speed of the lifting platform, shall not be, greater than 0,15 m/s$\rangle$.

2. Transformation des assertions LN en assertions RAINS

2.a Détection des syntagmes. L'exigence comporte une seule assertion dont les syntagmes sont :

1. **NP** (DT The) (VBN rated) (NN speed)
2. **PP** (IN of)
3. **NP** (DT the) (VBG lifting) (NN platform)
4. **VP** (MD shall) (RB not) (VB be)
5. **NP** (JJR greater) (IN than) (CD 0,15) (NN m) (NN /)
6. **VP** (VBZ s)

2.b Détection des entités hors ontologie de domaine. Les entités hors-ontologie de domaine (comparateurs, marque de négation, marqueurs de restrictions universelles, littéraux) sont les suivantes - de couleur bleue :

1. **NP** (DT The) (VBN rated) (NN speed)
2. **PP** (IN of)
3. **NP** (DT the) (VBG lifting) (NN platform)
4. **VP** (MD shall) not_{N} (VB be)
5. **NP** $\text{greater than}_{\text{K}}$ $0{,}15_{\text{F}}$ (NN m) (NN /)
6. **VP** (VBZ s)

Ces entités suggèrent un embryon de configuration qui est : ***N*KF***

2.c Identification des configurations RAINS La liste des configurations RAINS compatibles avec le motif ***N*KF*** est : $\mathcal{L} = \{\textbf{CDNKF}, \textbf{DCNKF}, \textbf{CODNKF}\}$

2.d Identification des entités de l'ontologie de domaine.

— Élagages syntaxique et sémantique

1. **NP** (~~DT The~~) (VBN rated) (NN speed) **#Élagage syntaxique**
2. ~~**PP (IN of)**~~ **#Type de syntagme non considéré**
3. **NP** (~~DT the~~) (VBG lifting) (NN platform) **#Élagage syntaxique**
4. **VP** (MD ~~shall~~) not_{N} (VB ~~be~~) **#Élagage syntaxique**
5. **NP** $\text{greater than}_{\text{K}}$ $0{,}15_{\text{F}}$ (NN ~~m~~) (NN ~~/~~) **#Élagage sémantique**
6. **VP** (VBZ ~~s~~) **#Élagage sémantique**

— Identification des entités proprement dites

1. "rated speed" $\rightarrow$ $\{\texttt{:ratedSpeed}_{\text{D}}\}$
2. "lifting platform" $\rightarrow$ $\{\texttt{:LiftingPlatform}_{\text{C}}, \texttt{:LIFTINGGEAR}_{\text{I}}\}$
3. Syntagme verbal vide $\rightarrow$ $\{\texttt{:undeterminedProperty}_{\text{X}}\}$
4. Syntagme nominal vide $\rightarrow$ { }
5. Syntagme verbal vide $\rightarrow$ $\{\texttt{:undeterminedProperty}_{\text{X}}\}$

— Liste des assertions RAINS candidates :

1. (ρ_0) "rated speed" "Lifting platform" "not" "undetermined property" "greater than" "0.15" "undetermined property"
   Configuration : **DCNXKFX**
2. (ρ_1) "rated speed" "LIFTING_GEAR" "not" "undetermined property" "greater than" "0.15" "undetermined property"
   Configuration : **DINXKFX**

2.e Résolution de l'assertion RAINS.

1. Pertinence des *undetermined properties*
   (a) (ρ_0) : **DCNXKFX** $\rightarrow$ **DCNKF** (car présence d'une *datatype property*). (ρ_0) devient
   "rated speed" "Lifting platform" "not" "greater than" "0.15"
   (b) (ρ_1) : **DINXKFX** $\rightarrow$ **DCNXKFX** (idem), d'où
   (ρ_1) "rated speed" "LIFTING_GEAR" "not" "greater than" "0.15"
2. Résolution de l'assertion RAINS proprement dite
   La configuration de (ρ_0) appartient à $\mathcal{L}$ et (ρ_0) est valide. En tant que unique membre de la catégorie *LV*, (ρ_0) est l'assertion finale.

3. Résolution de l'antécédent

3.a Prédicat principal. Le prédicat principal est l'auxiliaire ***be***.

3.b Négation du prédicat principal (ρ_0) devient
"rated speed" "Lifting platform" ~~"not"~~ "greater than" "0.15"

3.c Combinaison des assertions Au terme de cette étape, on obtient l'antécédent :
If "rated speed" "Lifting platform" "greater than" "0.15"

4. Résolution du conséquent

4.a Concept principal. Le concept principal est "Lifting platform"

4.b Détermination de la référence du concept principal. "Lifting platform" est la 2^e entité de l'antécédent. Le conséquent est : Then it is "Non-compliant"2
Règle RAINS finale :
If "rated speed" "Lifting platform" "greater than" "0.15"
Then it is "Non-compliant"2

Exemple 2 : ***"In buildings with public access, the platform length shall not be less than 1400 mm, to enable sufficient space for an attendant."***

1. Identification des assertions

Deux triplets sont extraits de cette exigence :
$(\tau_0)\langle$`the platform length, shall not be, less than 1400 mm to enable sufficient space for an attendant`$\rangle$
$(\tau_1)\langle$`In buildings with public access`$\rangle$

2. Transformation des assertions LN en assertions RAINS

2.a Détection des syntagmes.
Pour (τ_0)

1. **NP** (DT The) (NN platform) (NN length)
2. **VP** (MD shall) (RB not) (VB be)
3. **NP** (JJR less) (IN than) (CD 1400) (NN mm)
4. **VP** (TO to) (VB enable)
5. **NP** (JJ sufficient) (NN space)
6. **PP** (IN for)
7. **NP** (DT an) (NN attendant)

Pour (τ_1)

1. **PP** (IN In)
2. **NP** (NNS buildings)
3. **PP** (IN with)
4. **NP** (JJ public) (NN access)

2.b Détection des entités hors ontologie de domaine.
Pour (τ_0)

1. **NP** (DT The) (NN platform) (NN length)
2. **VP** (MD shall) $\text{not}_{\mathbf{N}}$ (VB be)
3. **NP** $\text{less than}_{\mathbf{K}}$ $1400_{\mathbf{i}}$ (NN mm)
4. **VP** (TO to) (VB enable)
5. **NP** (JJ sufficient) (NN space)
6. **PP** (IN for)
7. **NP** (DT an) (NN attendant)

Embryon de configuration : ***N*Ki***

Pour (τ_1)

Aucune entité hors ontologie de domaine.

Embryon de configuration : * - c'est-à-dire toute combinaison ne contenant pas d'entités hors-ontologie

2.c Identification des configurations RAINS. Pour (τ_0), la liste des configurations RAINS est $\mathcal{L}_0$ = {**CDNKi**, **DCNKi**, **CODNKi**} et pour (τ_1), on a $\mathcal{L}_1$ = {**I**, **BC**, **CC**, **C**, **COC**, **CB**, **COB**, **COI**}

2.d Identification des entités de l'ontologie de domaine.

Pour (τ_0)

— Élagages syntaxique et sémantique

1. **NP** (~~DT the~~) (NN platform) (NN length) **#Élagage syntaxique**
2. **VP** (MD ~~shall~~) not$_{N}$ (VB ~~be~~) **#Élagage syntaxique**
3. **NP** less than$_{K}$ 1400$_{i}$ (NN ~~mm~~) **#Élagage sémantique**
4. **VP** (TO ~~to~~) (VB ~~enable~~) **#Élagages syntaxique et sémantique**
5. **NP** (VB ~~sufficient~~) (NN space) **#Élagage sémantique**
6. ~~**PP** (IN for)~~ **#Type de syntagme non considéré**
7. **NP** (DT ~~an~~) (NN attendant) **#Élagage syntaxique**

— Identification des entités proprement dites

1. Pour rappel, on recherche les entités dans tous les ***sous-syntagmes nominaux***
   "platform" → {:LiftingPlatform$_{C}$}
   "length" → {:length$_{D}$, :ylength$_{O}$, :xlength$_{O}$}
   "platform length" → {:length$_{D}$, :y length$_{O}$, :x length$_{O}$}
   #Le syntagme "platform length" peut être formalisé soit comme 2 entités, identifiées à partir des 2 sous-syntagmes "platform" et "length", soit comme une entité identifiée à partir de l'expression "platform length"
2. Syntagme verbal vide → {:undeterminedProperty$_{X}$}
3. Syntagme nominal vide → { }
4. Syntagme verbal vide → {:undeterminedProperty$_{X}$}
5. "space" → { :IfcSpace$_{C}$, :IfcSpaceType$_{C}$, :IfcSpaceHeat$_{C}$}
6. "attendant" → {:ATTENDANCE$_{I}$}

— Liste des assertions RAINS candidates ("X" = "undetermined property") :

1. (ρ_0) "Lifting platform" **"length"** "not" "X" "less than" "1400" "X" "Ifc space" "ATTENDANCE"
   Configuration : **CDNXKiXCI**

2. (ρ_1) "Lifting platform" **"x length"** "not" "X" "less than" "1400" "X" "Ifc space" "ATTENDANCE"
   Configuration : **CONXKiXCI**
3. (ρ_2) "Lifting platform" **"y length"** "not" "X" "less than" "1400" "X" "Ifc space" "ATTENDANCE"
   Configuration : **CONXKiXCI**
4. (ρ_3) "length" "not" "X" "less than" "1400" "X" "Ifc space" "ATTENDANCE"
   Configuration : **DNXKiXCI**
5. (ρ_4) "x length" "not" "X" "less than" "1400" "X" "Ifc space" "ATTENDANCE"
   Configuration : **ONXKiXCI**
6. (ρ_5) "y length" "not" "X" "less than" "1400" "X" "Ifc space" "ATTENDANCE"
   Configuration : **ONXKiXCI**
7. (ρ_6) à (ρ_{11}) identiques à (ρ_0) à (ρ_5) avec "Ifc space type" à la place de "Ifc space"
8. (ρ_{12}) à (ρ_{17}) identiques à (ρ_0) à (ρ_5) avec "Ifc space heat" à la place de "Ifc space"

Pour (τ_1)

— Élagages syntaxique et sémantique

1. ~~**PP (IN In)**~~ **#Type de syntagme non considéré**
2. **NP** (NNS buildings)
3. ~~**PP (IN with)**~~ **#Type de syntagme non considéré**
4. **NP** (JJ public) (NN access)

— Identification des entités proprement dites

1. "buildings" → {:BUILDING$_{\mathbf{I}}$, :IfcBuilding$_{\mathbf{C}}$, :BUILDING_OWNER$_{\mathbf{I}}$}
2. "public access" → { :PublicAccess$_{\mathbf{C}}$, :publiclyAccessible$_{\mathbf{B}}$, :ACCESS$_{\mathbf{I}}$}

— Liste des assertions RAINS candidates :

1. (r_0) "BUILDING" "Public access"
   Configuration : **IC**
2. (r_1) "BUILDING" "publicly accessible"
   Configuration : **IB**
3. (r_2) "BUILDING_OWNER" "ACCESS"
   Configuration : **II**
4. (r_3) "BUILDING_OWNER" "Public access"
   Configuration : **IC**
5. (r_4) "BUILDING_OWNER" "publicly accessible"
   Configuration : **IB**

6. (r_5) `"BUILDING_OWNER" "ACCESS"`
   Configuration : **II**
7. (r_6) `"Ifc Building" "Public access"`
   Configuration : **CC**
8. (r_7) `"Ifc Building" "publicly accessible"`
   Configuration : **CB**
9. (r_8) `"Ifc Building" "ACCESS"`
   Configuration : **CI**

2.e Résolution de l'assertion RAINS.

Pour (τ_0)

1. Pertinence des *undetermined properties*
   Toutes les assertions RAINS candidates possèdent une *datatype* ou une *object property*. Les propriétés indéterminées sont ainsi ignorées. On a donc
   (a) (ρ_0) : **CDNXKiXCI** $\rightarrow$ **CDNKiCI**, d'où
   `"Lifting platform" "length" "not" "less than" "1400" "Ifc space" "ATTENDANCE"`
   (b) (ρ_1) : **CONXKiXCI** $\rightarrow$ **CONKiCI**, d'où
   `"Lifting platform" "x length" "not" "less than" "1400" "Ifc space" "ATTENDANCE"`
   (c) Idem (retrait de `"X"`) pour (ρ_2) à (ρ_{17})
2. Résolution de l'assertion RAINS proprement dite
   La liste restreinte des configurations RAINS de (τ_0) est $\mathcal{L}_0 = \{$**CDNKi**, **DCNKi**, **CODNKi**$\}$. Aucune des configurations des assertions candidates (c'est-à-dire (ρ_0) à (ρ_{17})) ne figure dans $\mathcal{L}_0$. On se retrouve donc avec pour seule catégorie $\overline{L}\overline{V}$ qui contient toutes les configurations (ρ_0) - (ρ_{17}). Comme le montre le tableau ci-dessous, qui présente le vote des configurations des assertions de $\overline{L}\overline{V}$ en faveur des configurations de $\mathcal{L}_0$, la configuration **CDNKi** est majoritairement plébiscitée (boucle ❼ de l'algorithme B.1, page 190).

Configurations de $\mathcal{L}_0$	ρ_0, ρ_6, ρ_{12} (**CDNKiCI**)	ρ_1, ρ_2, ρ_7, ρ_8, ρ_{13}, ρ_{14} (**CONKiCI**)	ρ_3, ρ_9, ρ_{15} (**DNKiCI**)	ρ_4, ρ_5, ρ_{10}, ρ_{11}, ρ_{16} et ρ_{17} (**ONKiCI**)
CDNKi	✓	✓	✓	✓
DCNKi	✗	✗	✓	✓
CODNKi	✗	✗	✗	✗

On a donc $\mathcal{L}'_0 = \{$**CDNKi**$\}$. Par ailleurs, de toutes les configurations de $\overline{L}\overline{V}$ plébiscitant CDNKi, **CDNKiCI** est celle qui est la plus proche avec une distance de Levenshtein de 2 (contre 3 et 4 pour les autres) de cette dernière. CDNKiCI est

donc la `candidateConfig`. Ensuite on trie les assertions RAINS ayant pour configuration la `candidateConfig`. Ce sont (ρ_0), (ρ_6) et (ρ_{12}). Un tri décroissant de ces assertions permet d'obtenir la liste ordonnée $\{$ (ρ_0), (ρ_6), (ρ_{12})$\}$. En prenant le premier élément de cette liste (c'est-à-dire (ρ_0)), on effectue une itération sur $\mathcal{L}'_0$. Cette itération prend fin dès lors qu'on a pu transformer (ρ_0) en une assertion de configuration autorisée ou lorsqu'on arrive au terme de $\mathcal{L}'_0$. Pour transformer (ρ_0) de configuration **CDNiCI** en une assertion de configuration (ρ_0), on s'appuie sur la séquence d'édition **{None, None, None, None, None, Delete, Delete}**[15]. Cette séquence impose de supprimer les 2 derniers caractères de **CDNKiCI** et donc de se séparer du concept et de l'individu présents aux dernières positions de (ρ_0). En écartant les entités codant pour le **C** et **I** de **CDNKiCI**, (ρ_0) devient :
`"Lifting platform" "length" "not" "less than" "1400"`
En plus d'avoir une configuration RAINS autorisée, cette assertion est sémantiquement valide. En effet, `:LiftingPlatform` $\in$ `domain(:length)` et `:length` $\in$ `in-properties(xsd:integer)`. C'est donc la version RAINS de (τ_0).

Pour (τ_1)

1. Pertinence des *undetermined properties*
   Il n'y a pas de propriétés indéterminées dans les assertions candidates
2. Résolution de l'assertion RAINS proprement dite
   La liste restreinte des configurations est $\mathcal{L}_1 = \{\mathbf{I}, \mathbf{BC}, \mathbf{CC}, \mathbf{C}, \mathbf{COC}, \mathbf{CB}, \mathbf{COB}, \mathbf{COI}\}$. Le partitionnement des assertions candidates donne : $L\overline{V} = \{(r_6),\ (r_7)\}$ et $\overline{L}\overline{V} = \{(r_0), (r_1), (r_2), (r_3), (r_4), (r_5), (r_8)\}$. On utilisera donc les assertions de la catégorie $L\overline{V}$ à savoir (r_6) et (r_7). Pour départager (r_6) et (r_7), on les range par score de similarité sur les concepts et c'est (r_6) qui possède un meilleur score. (r_6) est une double concept assertion et les double concept assertion sont par définition toujours valide. La version RAINS de (τ_1) est :
   (r_6) `"Ifc Building" "Public access"`

3. Résolution de l'antécédent

3.a Prédicat principal. Le prédicat principal est l'auxiliaire ***be***.

3.b Négation du prédicat principal (τ_0) est la seule assertion en LN faisant usage du prédicat principal. Par conséquent, seule l'assertion RAINS dérivée de (τ_0) est concernée par cette tâche qui a pour résultat :
(ρ_0) `"Lifting platform" "length"` ~~`"not"`~~ `"less than" "1400"`

3.c Combinaison des assertions Au terme de cette étape, on obtient l'antécédent :
`Given "Lifting platform" "length" "less than" "1400"`

15. Pour rappel **None** = pas de changement et **Delete** = suppression du caractère. Voir algorithme B.3.

L'assertion (r_6) est écartée parce qu'elle n'a pas de lien (concept en commun), avec (ρ_0) - voir la ligne ❸ de l'algorithme 5.2.

4. Résolution du conséquent

4.a Concept principal. Le concept principal est `"Lifting platform"`

4.b Détermination de la référence du concept principal. `"Lifting platform"` est la première entité de l'antécédent. Le conséquent est : `Then it is "Non-compliant"1`
Règle RAINS finale :

```
Given "Lifting platform" "length" "less than" "1400"
Then it is "Non-compliant"1
```

Exemple 2' : *"In public buildings, the platform length shall not be less than 1400 mm, to enable sufficient space for an attendant."*

Cet exemple ***fictif*** (la règle n'est pas mentionnée telle qu'elle dans le texte normatif) nous permet d'illustrer le mécanisme de détection des annotations dans FORSA. L'exemple reprend presque la totalité de la règle de l'exemple précédent en changeant l'expression "***In buildings with public access***" par "***In public buildings***". En déroulant cet exemple, nous ferons mention de certains traitements équivalents de l'exemple précédent pour éviter la répétition. Cette mention sera signalée par le terme **Idem**.

1. Identification des assertions

Deux triplets sont extraits de cette exigence :
(τ_0)⟨`the platform length, shall not be, less than 1400 mm to enable sufficient space for an attendant`⟩
(τ_1)⟨`In public buildings`⟩

2. Transformation des assertions LN en assertions RAINS

2.a Détection des syntagmes.
Pour (τ_0)

Idem

Pour (τ_1)

1. **PP** (IN In)
2. **NP** (JJ public) (NNS buildings)

2.b Détection des entités hors ontologie de domaine.
Pour (τ_0)

Idem

Pour (τ_1)
Aucune entité hors ontologie de domaine.
Embryon de configuration : * - c'est-à-dire toute combinaison ne contenant pas d'entités hors-ontologie
2.c Identification des configurations RAINS. Pour (τ_0), la liste des configurations RAINS est $\mathcal{L}_0 = \{$**CDNKi**, **DCNKi**, **CODNKFi**$\}$ et pour (τ_1), on a $\mathcal{L}_1 = \{$**I**, **BC**, **CC**, **C**, **COC**, **CB**, **COB**, **COI**$\}$
2.d Identification des entités de l'ontologie de domaine.
Pour (τ_0)

Idem

Pour (τ_1)

— Élagages syntaxique et sémantique

1. ~~**PP (IN In)**~~ `#Type de syntagme non considéré`
2. **NP** (JJ public) (NNS buildings)

— Identification des entités proprement dites

1. "public buildings" $\rightarrow$ {`:PUBLIC_BUILDING`$_{\mathbf{I}}$, `:BUILDING`$_{\mathbf{I}}$, `:IfcBuilding`$_{\mathbf{C}}$}

— Liste des assertions RAINS candidates :

1. (r_0) `"PUBLIC_BUILDING"`
   Configuration : **I**
2. (r_1) `"BUILDING"`
   Configuration : **I**
3. (r_2) `"IfcBuilding"`
   Configuration : **C**

2.e Résolution de l'assertion RAINS.
Pour (τ_0)

Idem. On aboutit à :
(ρ_0) : `"Lifting platform" "length" "not" "less than" "1400"`

Pour (τ_1)

1. Pertinence des *undetermined properties*
   Il n'y a pas de propriétés indéterminées dans les assertions candidates
2. Résolution de l'assertion RAINS proprement dite
   La liste restreinte des configurations est $\mathcal{L}_1 = \{\mathbf{I}, \mathbf{BC}, \mathbf{CC}, \mathbf{C}, \mathbf{COC}, \mathbf{CB}, \mathbf{COB}, \mathbf{COI}\}$.
   Le partitionnement des assertions des candidates donne : $LV = \{(r_0)\}$ et $L\overline{V} = \{(r_1), (r_2)\}$. LV étant non vide, c'est de là que proviendra l'assertion RAINS finale.

(r_0), de configuration **I**, étant l'unique assertion en lice, c'est elle l'assertion finale. Notons que r_0 est valide car elle est formée d'une seule entité qui est un individu. De plus, cet individu est une valeur possible d'annotation car la requête ci-dessous renvoie une valeur qui est `:buildingUse`

```
SELECT ?annotationName
WHERE { ?annotationName rdfs:domain ?annotation.
        ?annotation rdfs:subClassOf* :Annotation.
        ?annotationName rdfs:range ?enum.
        :PUBLIC_BUILDING rdf:type ?type.
        ?type rdfs:subClassOf* ?enum. }
LIMIT 1
```

(r_0) est donc transformée en une annotation qui est :
`@Building use (Public building)`

3. Résolution de l'antécédent

3.a Prédicat principal. Le prédicat principal est l'auxiliaire ***be***.

3.b Négation du prédicat principal. (τ_0) est la seule assertion en LN faisant usage du prédicat principal. Par conséquent, seule l'assertion RAINS dérivée de (τ_0) est concernée par cette tâche qui a pour résultat :
(ρ_0) `"Lifting platform" "length" ~~"not"~~ "less than" "1400"`

3.c Combinaison des assertions. Au terme de cette étape, on obtient l'antécédent :
`@Building use (Public building)`
`Given "Lifting platform" "length" "less than" "1400"`
L'assertion (r_0), en tant qu'annotation est placée au début de l'antécédent, comme l'exige la syntaxe de RAINS (voir bloc de lignes ❶ de l'algorithme 5.2).

4. Résolution du conséquent

4.a Concept principal. Le concept principal est `"Lifting platform"`

4.b Détermination de la référence du concept principal. `"Lifting platform"` est la première entité de l'antécédent (les entités des annotations ne sont pas considérées dans ce décompte - voir bloc de lignes ① de l'algorithme 5.3). Le conséquent est : `Then it is "Non-compliant"1`
Règle RAINS finale :

```
@Building use (Public building)
Given "Lifting platform" "length" "less than" "1400"
Then it is "Non-compliant"1
```

Exemple 3 : ***"For doors with multiple panels, one of the panels must have a minimal width of 0.80 meter"***

1. Identification des assertions

Deux triplets sont extraits de cette exigence :
(τ_0) ⟨`one of the panels, must have, a minimal width of 0.80 meter`⟩
(τ_1) ⟨`For doors with multiple panels`⟩

2. Transformation des assertions LN en assertions RAINS

2.a Détection des syntagmes.

Pour (τ_0)

1. **NP** (CD one)
2. **PP** (IN of)
3. **NP** (DT the) (NNS panels)
4. **VP** (MD must) (VB have)
5. **NP** (DT a) (JJ minimal) (NN width)
6. **PP** (IN of)
7. **NP** (CD 0.80) (NN meter)

Pour (τ_1)

1. **PP** (IN For)
2. **NP** (NNS doors)
3. **PP** (IN with)
4. **NP** (JJ multiple) (NNS panels)

2.b Détection des entités hors ontologie de domaine.

Pour (τ_0)

1. **NP** one_i
2. **PP** (IN of)
3. **NP** (DT the) (NNS panels)
4. **VP** (MD must) (VB have)
5. **NP** (DT a) (JJ minimal) (NN width)
6. **PP** (IN of)
7. **NP** 0.80_F (NN meter)

Embryon de configuration : ***Ki*KF***

Pour (τ_1)

Aucune entité hors ontologie de domaine.
Embryon de configuration : * .

2.c Identification des configurations RAINS.

Pour (τ_0), la liste des configurations RAINS est $\mathcal{L}_0 = \emptyset$ car aucun élément de l'ensemble possible des configuration RAINS ne correspond au fragment de configuration ***Ki*KF***. Pour (τ_1), on a $\mathcal{L}_1 = \{$**I**, **BC**, **CC**, **C**, **COC**, **CB**, **COB**, **COI**$\}$. $\mathcal{L}_0$ ***étant vide, on écarte*** (τ_0) ***de la suite des traitements***.

2.d Identification des entités de l'ontologie de domaine.

Pour (τ_1)

— Élagages syntaxique et sémantique

1. ~~**PP** (IN For)~~ `#Type de syntagme non considéré`
2. **NP** (NNS doors)
3. ~~**PP** (IN with)~~ `#Type de syntagme non considéré`
4. **NP** (JJ multiple) (NN panels)

— Identification des entités proprement dites

1. "doors" $\rightarrow$ {`:IfcDoor`$_{\mathbf{C}}$, `:TRAP_DOOR`$_{\mathbf{I}}$, `:DOOR`$_{\mathbf{I}}$}
2. "multiple panels" $\rightarrow$ {`:PANEL`$_{\mathbf{I}}$}

— Liste des assertions RAINS candidates :

1. (r_0) `"IfcDoor" "PANEL"`
   Configuration : **CI**
2. (r_1) `"TRAP_DOOR" "PANEL"`
   Configuration : **II**
3. (r_2) `"DOOR" "PANEL"`
   Configuration : **II**

2.e Résolution de l'assertion RAINS.

Pour (τ_1)

1. Pertinence des *undetermined properties* Il n'y a pas de propriétés indéterminées dans les assertions candidates
2. Résolution de l'assertion RAINS proprement dite
   La liste restreinte des configurations RAINS de (τ_1) est $\mathcal{L}_1 = \{$**I**, **BC**, **CC**, **C**, **COC**, **CB**, **COB**, **COI**$\}$. Aucune des configurations des assertions candidates ne figure dans $\mathcal{L}_1$ donc on va devoir procéder à un vote.

Configurations de $\mathcal{L}_1$	r_0 **(CI)**	r_1, r_2 **(II)**
I	✓	✓
BC	✗	✗
CC	✓	✗
CB	✓	✗
COB	✗	✗
C	✓	✗
COC	✗	✗
COI	✓	✗

Le tableau ci-dessus montre que **I** est la configuration de $\mathcal{L}_1$ la plus plébiscitée. On a donc $\mathcal{L}'_1 = \{\mathbf{I}\}$. De plus, les configurations **CI** et **II** sont à égale distance de la configuration **I**. En outre, aucune n'utilise de propriétés. Pour départager ces configurations, FORSA calcule, pour les assertions d'une configuration candidate donnée, la moyenne des scores de similarité des concepts de ces assertions. La configuration candidate finale est celle ayant la plus grande moyenne. Dans cet exemple, c'est la configuration *CI* qui est privilégiée par rapport à **II** (le *matching score* de (r_0) est supérieur à la moyenne des *matching scores* de (r_1) et (r_2)). Enfin, on effectue une itération sur $\mathcal{L}'_1$ jusqu'à obtenir une modification autorisée de la configuration **CI** selon les éléments de $\mathcal{L}'_1$. $\mathcal{L}'_1$ ne comportant qu'un élément, il y a une seule itération a effectuer. Au cours de cette itération, on calcule la séquence d'édition de **CI** en **I** et on obtient {**Delete, None**}. Il faut donc supprimer le concept dénoté par la **C** de **CI** et conserver l'individu référencé par le *I* de **CI**. (r_0) est donc réduit à une assertion RAINS formée d'une seule entité qui est `"PANEL"`.

(r_0) étant devenu une assertion mono-entité et composé de l'individu `"PANEL"`, on vérifie que `:PANEL` est candidat à une valeur d'annotation. Cette vérification est effectuée à l'aide de la requête ci-dessous.

```
SELECT ?annotationName
WHERE { ?annotationName rdfs:domain ?annotation.
        ?annotation rdfs:subClassOf* :Annotation.
        ?annotationName rdfs:range ?enum.
        :PANEL rdf:type ?type.
        ?type rdfs:subClassOf* ?enum. }
LIMIT 1
```

Cette requête ne renvoie aucun résultat. (r_0) ne peut donc pas devenir une annotation et est écartée.

Ceci a pour conséquence l'abandon de l'assertion-LN (τ_1) dans le processus de formalisation.

(τ_0) et (τ_1) étant écartées, le processus de formalisation est interrompu et un message d'erreur est renvoyé.

5.6 Comparaison entre FORSA et les approches de l'état de l'art

À la section 3.3.2, nous avons présenté des méthodes de formalisation automatique de règles. Nous avons mentionné le fait que le matériel de test de ces outils n'étaient pas disponible de même que les implémentations de ces méthodes. Cela rend impossible une comparaison chiffrée entre ces approches et FORSA. Toutefois, nous résumons un comparatif entre ces méthodes et FORSA au travers du tableau 5.3 ci-dessous.
Pour rappel, on a (voir section 3.3 pour plus de détails) :

($\mathcal{F}_1$) Morcellement d'une exigence complexe en plusieurs exigences élémentaires ;
($\mathcal{F}_2$) Détermination du contexte d'application des règles ;
($\mathcal{F}_3$) Gestion des différences entre les termes des textes et leur normalisation dans le vocabulaire de domaine ;
($\mathcal{F}_4$) Résolution des paraphrases
($\mathcal{F}_5$) Résolution des co-références et des anaphores
($\mathcal{F}_6$) Détermination de l'antécédent et du conséquent de la règle
($\mathcal{F}_7$) Mise en évidence des paramètres implicites
($\mathcal{F}_8$) Élagage

5.7 Conclusion du chapitre

Au cours du chapitre 4, nous avons soulevé la problématique de la non-autonomie des experts métiers pour la constitution de dépôts de normes formelles ; ces derniers devant nécessairement se faire assister par des logiciens pour cette tâche. Pour permettre aux experts métiers d'être les seuls opérateurs oeuvrant à la mise sur pied et à la gestion d'un corpus de règles formelles, nous avons proposé la manipulation des règles métiers à travers le langage RAINS. Toutefois, au regard de la taille des corpus technico-réglementaires, la réécriture manuelle de chaque exigence en règle RAINS reste une opération gourmande en ressources. Ainsi, nous nous sommes fixés pour objectif, de proposer un service offrant la réécriture automatique d'exigences réglementaires sous la forme d'une phrase RAINS. Ce service implémente une nouvelle approche dénommée FORSA. La règle RAINS produite par ce service est soumis à l'appréciation de l'expert métier qui devra y apporter des modifications si nécessaire.
FORSA est une méthode de formalisation qui prend en entrée une prescription réglementaire, sous la forme d'une ***phrase*** et réécrit automatiquement celle-ci sous la forme d'une règle RAINS. RAINS est un langage contrôlé dont l'un des objectifs principaux est de transformer automatiquement toute phrase RAINS en une expression formelle. Pour cela, RAINS s'appuie sur une ***ontologie*** conceptualisant le domaine de connaissance duquel est issu la phrase RAINS. Cette ontologie est supposée existante et constitue le

Tableau 5.3 – Comparaison entre FORSA et les approches de l'état de l'art.

Caractéristiques	**Hassanpour *et al.* [2011]**	**Kang *et al.* [2015]**	**FORSA**
Entrées	Phrase en LN Ontologie Quelques règles formelles	Texte en LN Ontologie -	Phrase en LN Ontologie -
Sortie	Règle SWRL	Règle SADL	Règle RAINS (règle SPARQL)
Usage des règles en sortie	Non définie	Non définie	Détection de non-conformité
Mesure de similarité	Needleman & Wunsch	-	Cosinus + Jaccard_2
Détection d' *object properties*	Oui	Oui	Oui
Détection de *datatype props*	Non	Oui	Oui
Détection d'annotations	Non	Non	Oui
Fonctionnalités			
($\mathcal{F}_1$)	✓	✗	✗
($\mathcal{F}_2$)	✗	✗	✓
($\mathcal{F}_3$)	✓	✗	✓
($\mathcal{F}_4$)	✓	✗	✗
($\mathcal{F}_5$)	✗	✓	✗
($\mathcal{F}_6$)	✓	✓	✓
($\mathcal{F}_7$)	✗	✗	✗
($\mathcal{F}_8$)	✗	✗	✓

second paramètre d'entrée de FORSA.

Le fonctionnement proprement dit de la méthode FORSA comporte quatre grandes étapes. La première grande étape est le morcellement de l'exigence en un ensemble d'assertion atomiques. Le but de cette étape est de disposer de fragments autonomes de la phrase et ainsi de faire un grand pas vers la syntaxe du langage RAINS qui impose de décomposer les exigences en des assertions RAINS atomiques. La deuxième grande étape consiste à transformer chacune des assertions atomiques obtenues précédemment, en des assertions RAINS ou en des annotations. Il est important de noter que dans certaines circonstances, des assertions en langages naturelles peuvent être considérés comme sémantiquement non pertinentes et ignorées de la suite des traitements. La troisième étape importante consiste à réunir toutes les assertions et annotations obtenues précédemment pour constituer l'antécédent de la règle. Au cours de cette opération, on distingue un mécanisme sous-jacent important que nous appelons ***négation du prédicat principal***. Le prédicat principal est le verbe autour duquel est construit l'exigence.

FORSA s'appuie sur ce dernier afin que la règle RAINS finale soit une règle permettant de s'attaquer uniquement aux objets présentant des défauts. Il peut arriver que le prédicat principal se trouve dans une assertion ayant été élaguée dans les étapes précédentes. Dans ce cas, le processus de formalisation s'interrompt et FORSA renvoie un message d'erreur. La quatrième et dernière étape de FORSA est la résolution du conséquent de la règle. Elle consiste dans un premier temps à identifier le conséquent principal de la règle et ensuite à trouver sa position dans l'antécédent ; le numéro de cette position servira de co-référence entre l'entité dite de non-conformité du conséquent et le concept principal. Ce dernier représente le concept de l'antécédent qui a été obtenu à partir du sujet du prédicat principal de l'exigence.
Le chapitre suivant est consacré aux évaluations de RAINS et de FORSA. Nous présentons le matériel d'évaluation, les résultats de cette évaluation ainsi qu'une analyse critique de chacune de ces contributions.

Chapitre 6

Évaluation et Discussion

Contents

Dans le chapitre 4, nous avons présenté le langage naturel contrôlé RAINS, qui permet de représenter les normes sous une forme semi-formelle. Au chapitre 5 nous avons exposé l'approche FORSA permettant de transformer une norme du langage naturel en une règle RAINS. Pour chacune de ces deux contributions, nous avons effectué un ensemble de tests permettant d'évaluer la prise en main, dans le cas du langage RAINS, l'efficacité globale, en ce qui concerne l'approche FORSA. Dans ce chapitre, nous présentons les données de tests, le protocole d'évaluation, les résultats de cette évaluation. Aussi, nous effectuons une analyse critique de ces résultats.

6.1 IfcOWL pour les plate-formes élévatrices

À la section 3.2.1 (pages 38 et suivantes), nous avons introduit l'ontologie IfcOWL. C'est une représentation du standard IFC (voir section 2.2.2) sous la forme d'une ontologie de domaine en OWL. Cette ontologie nous permet de tirer profit des des standards du Web Sémantique (interopérabilité, partage des données, mécanismes d'inférences, équilibre entre expressivité et passage à l'échelle) tout en s'exprimant dans un vocabulaire compréhensible par les experts métiers du domaine de la Construction. Plusieurs versions OWL des IFC existent dans la littérature. Un état de l'art très récent de ces différents IfcOWL est proposé dans le travail de [Pauwels & Terkaj 2016]. Dans ce manuscrit, nous ne refaisons pas de comparaisons et nous appuyons sur les conclusions de Pauwels & Terkaj [2016]. Dans leurs travaux, Pauwels & Terkaj proposent eux-aussi une nouvelle version de IfcOWL. C'est sur cette dernière que nous nous sommes appuyés pour les divers exemples qui illustrent nos contributions et pour bâtir nos données de tests. Contrairement aux précédentes versions de IfcOWL, celle de Pauwels & Terkaj a un double avantage :

(i) Dans les conversions IFC (exprimé en EXPRESS) vers OWL, certains axiomes du langage EXPRESS ne sont pas souvent représentés, car complexes. Or dans la version IfcOWL de Pauwels & Terkaj, l'ensemble des axiomes EXPRESS convertis en OWL ***inclus strictement*** ceux convertis dans les précédentes versions. C'est donc une version d'IfcOWL plus proche des IFC natifs que ne le sont les précédentes versions.

(ii) Les précédentes versions d'IfcOWL ont une expressivité couverte par OWL 1 DL alors que la version de Pauwels & Terkaj dispose d'axiomes de OWL 2 DL. Or OWL 2 DL est plus expressive que OWL 1 DL tout en restant décidable. Par exemple, le premier comprend les restrictions de cardinalité qualifiées contrairement au second.

Dès les premiers exemples que nous avons fournis tout au long de ce manuscrit, nous avons mentionné le fait que nos illustrations s'appuient sur la norme [NF EN 81-41 2011]. C'est un texte réglementaire dictant les règles de sécurité pour la construction et

l'installation des plates-formes élévatrices. Le vocabulaire employé dans ce texte réglementaire comportent des termes qui ne se trouvent pas de manière native dans les IFC et qui, par conséquent, sont absents de IfcOWL. Ainsi, pour être en mesure d'exprimer les prescriptions mentionnées dans cette norme, il est nécessaire d'enrichir IfcOWL avec des expressions issues du domaine des élévateurs.
Pour cela, nous nous sommes appuyés sur le glossaire proposé dans ce texte réglementaire proprement dit. En effet, la section 3 de la norme [NF EN 81-41 2011] on retrouve un glossaire composé de quarante-cinq (45) termes et de leur définitions en langage naturel. Comme l'indique la norme, ce glossaire est complété par celui de la norme ***EN ISO 12100-1 :2003*** [1]. En utilisant les éléments de ces glossaires, l'enrichissement d'IfcOWL a été effectué avec ***le support et la validation des experts du CSTB***.
Pour que l'ontologie IfcOWL que nous venons d'enrichir, soit utilisable pour représenter les exigences formelles dans leur environnement, il faut y ajouter des entités servant pour les annotations. En effet, lors de la présentation de l'organisation du corpus technico-réglementaire (voir section 2.1.2), nous avons souligné l'importance de la représentation des informations descriptives accompagnant les réglementations. Nous avons donné à ces descriptions le nom d'annotations. L'un des fruits des travaux de thèses de Yurchyshyna [2009] est la modélisation de ces annotations sous la forme d'une ontologie OWL. Cette ontologie porte le nom d'`annoComplexe.owl` et on peut voir un extrait de cette ontologie à la figure 4.2. Pour doter IfcOWL d'entités pouvant représenter des annotations, nous y avons importé l'ontologie `annoComplexe.owl`. Un extrait de cette ontologie IfcOWL deux fois enrichie a été présenté au chapitre 4 à travers la figure 4.1. Cette ontologie, pour laquelle nous avons conservé le nom IfcOWL tout au long de ce manuscrit, est celle qui a servi de support pour la formalisation des exigences extraites de la norme [NF EN 81-41 2011].

6.2 Évaluation de RAINS

Dans cette section, nous présentons les données et le protocole de test pour l'évaluation de la prise en main du langage RAINS.

6.2.1 Protocole d'évaluation de RAINS

Pour évaluer le langage RAINS, nous avons sélectionné de manière ***aléatoire***, ***cinquante*** (**50**) exigences de la norme [NF EN 81-41 2011]. Ces exigences avaient pour seules contraintes, de respecter la typologie des normes rentrant dans le cadre de notre travail (voir section 3.1.5) à savoir que ces exigences doivent être des phrases contenant des

1. Cette norme est aujourd'hui révisée par la norme ISO 12100 :2010 `http://www.iso.org/iso/fr/catalogue_detail.htm?csnumber=51528`

prescriptions strictes, sans références croisées ou implicites et complètement interprétables.
Pour chacune de ces 50 exigences, deux experts métiers ont proposé une version RAINS. Rappelons que chaque phrase RAINS peut être automatiquement transformée en une expression formelle - en requête SPARQL, pour être plus précis. Avant la réécriture des exigences en règles RAINS, nous avons effectué une présentation du langage RAINS aux experts métiers, illustrée par plusieurs exemples. Après qu'ils aient effectué leur travail de formalisation, nous avons recueilli leur retour d'expérience sur leur capacité à s'approprier le langage RAINS. La synthèse de leurs analyses est présentée à la section 6.2.2. En outre, nous avons relevé les erreurs présentes dans les différentes règles RAINS ; elles sont détaillées à la section 6.2.3. Ces erreurs ont été corrigées et nous disposons ainsi de 50 exigences en LN et de leur version RAINS. Ce corpus servira de référence pour l'évaluation de FORSA (voir section 6.3).

6.2.2 Retours d'expérience des experts métiers

Après que les experts métiers aient manipulé le langage RAINS au travers de plusieurs dizaines d'exemples concrets, il ressort que :

— La mise en évidence des entités RAINS à l'aide de guillemets n'est pas naturel mais demeure un moyen d'annotation facile et peu coûteux. Cette contrainte imposée par la syntaxe de RAINS est estimée faible parce qu'en plus des guillemets, le seul moyen de typer une entité est d'utiliser convenablement les lettres majuscules ou minuscules. C'est un mécanisme d'annotation qui demande moins d'effort que l'utilisation des balises traditionnelles (c'est-à-dire `<balise> texte </balise>`).
— Le découpage de la règle en plusieurs assertions est un mécanisme inhabituel bien qu'il permette de relever immédiatement les différentes conditions contenues dans une règle.
— La variété des types et sous-types d'assertions fait que même sans connaître par coeur la syntaxe formelle du langage, il n'est pas difficile d'écrire une phrase RAINS qui en définitive, s'avère syntaxiquement correcte.
— Le mécanisme de spécification des co-références n'est pas aisé à appliquer. Devoir déterminer manuellement le rang des entités entrave la fluidité du processus d'écriture d'une règle RAINS.
— La confusion entre des entités peut survenir lors de l'écriture d'une règle RAINS. En effet, connaître par coeur le schéma IFC et toutes ses entités (plus de 5000) et en écrivant la règle RAINS on peut mentionner une entité tout en ayant à l'esprit une autre.
— Une exigence peut conduire à plusieurs règles RAINS. Ces dernières peuvent se différencier par des tournures syntaxiques, par exemple l'inversion ou non de la position du concept et du prédicat dans une assertion. Rappelons que, dans

ce cas, les règles RAINS ont la même sémantique. Toutefois, il peut avoir des différences au niveau des entités employées pour semi-formaliser une exigence (et il est important de garder une trace des différentes versions semi-formelles de chaque exigence).

6.2.3 Résultats

Après avoir discuté du ressenti des experts métiers après une la manipulation de RAINS, nous présentons les types d'erreurs que l'on a retrouvés dans les règles RAINS produites par les experts avant correction. Ces erreurs sont comptabilisées dans le tableau 6.1.

Tableau 6.1 – Types d'erreurs commises lors de l'écriture de règles RAINS (le symbole # veut dire ***nombre***)

Types d'erreurs	**Expert 1** (**`# d'erreurs`** dans **`# de règles`**)	**Expert 2** (**`# d'erreurs`** dans **`# de règles`**)
Utilisation des guillemets	0 dans aucune règle	1 dans 1 règle
Agencement des entités	1 dans 1 règle	1 dans 1 règle
Distinction des types d'entités	0 dans aucune règle	0 dans aucune règle
Gestion des co-références	3 dans 2 règles	3 dans 2 règles
Relations entre entités	4 dans 4 règles	5 dans 4 règles

D'une part, on peut constater que l'assimilation par les experts, de la délimitation des entités par les guillemets, de l'agencement des entités et la distinction des types d'entités à l'aide des majuscules et minuscules est quasiment parfaite. D'autre part, la majorité des erreurs provient de la gestion des co-références et des relations sémantiques entre les entités de sorte qu'elles respectent le schéma de l'ontologie IfcOWL. Si l'on prend l'ensemble des 50 règles, le premier expert a commis des erreurs dans sept (7) règles et le second dans 7 règles aussi.

6.2.4 Analyse des résultats

Les erreurs que l'on retrouve dans les règles formelles écrites par les deux experts sont en adéquation avec les difficultés rencontrées par ceux-ci et que nous avons consignées à la section 6.2.2. En effet, les co-références se basent sur la position des entités. Or, il peut arriver de se tromper en dénombrant les entités pour trouver le rang de la position de l'entité à référencer. En ce qui concerne les relations entre entités, elle traduit le fait que dans certaines règles, les experts ont employé les prédicats inadéquats pour relier deux concepts ou un concept et un litéral. Cela n'est pas dû à une méconnaissance du schéma IfcOWL mais plutôt à des erreurs d'inattention se caractérisant par la confusion d'une propriété avec son inverse ou l'emploi d'une entité en ayant en l'idée une autre. Cette erreur reflète bien le besoin des experts d'un éditeur de règles RAINS qui propose

les entités candidates pour une écriture immédiate de règles syntaxiquement et sémantiquement valides. Aussi, cet éditeur de règles devra permettre d'obtenir sans difficulté les nombres permettant de référencer les concepts. De plus, il devra offrir la possibilité pour les experts, de fournir plusieurs versions RAINS de la même règle ; ces dernières devant être stockées dans le dépôt de règles et considérées comme des alternatives les unes des autres.

6.3 Évaluation de FORSA

À la section précédente, nous avons constitué, avec le concours des experts métiers, un dépôt de 50 exigences ainsi que de leurs versions RAINS. Rappelons ces règles RAINS s'appuient sur l'ontologie de domaine IfcOWL, telle que présentée à la section 6.1. Nous avons déroulé la méthode FORSA en prenant en entrée chacune de ces exigences ainsi que l'ontologie de domaine IfcOWL. Dans cette section, nous présentons les résultats de cette opération. Toutefois, avant de s'attarder sur les résultats obtenus par FORSA, nous définissons les métriques employées pour quantifier ces résultats.

6.3.1 Modélisation d'une règle RAINS

Nous avons relevé le fait qu'une exigence réglementaire peut être semi-formalisée par des règles RAINS ayant des syntaxes différentes ou employant des entités différentes. Ainsi, pour une exigence en langage naturel, il y a plusieurs règles RAINS possibles. Pour une exigence donnée, il est nécessaire de proposer une manière adéquate pour comparer le résultat fourni par une approche de formalisation et les résultats proposés par les experts. Pour cela, nous modélisons chaque règle RAINS du corpus de référence de manière à encapsuler toutes les représentations possibles d'une exigence donnée. Ladite modélisation est schématisée par la figure 6.1. Sur ce schéma, on peut voir que pour chaque exigence :

- On a sa version originale en langage naturel
- On dispose de la liste des annotations, s'il y en a. Chaque Annotation comporte un nom et la liste des valeurs correspondantes
- On a l'***ensemble des assertions qui composent les antécédents des règles RAINS pouvant représenter cette règle***. Chaque assertion se compose :
  - De zéro ou plusieurs entités non ontologiques (comparateurs, litéraux, marque de négation, marque de restriction universelle)
  - D'une ou plusieurs entités issues de l'ontologie de domaine
  - Des ***unités de sens*** présents dans l'exigence. C'est sur ces dernières que reposent principalement le mécanisme de gestion de la pluralité des entités pour la formalisation d'une même exigence (voir exemple ci-dessous pour illustration).

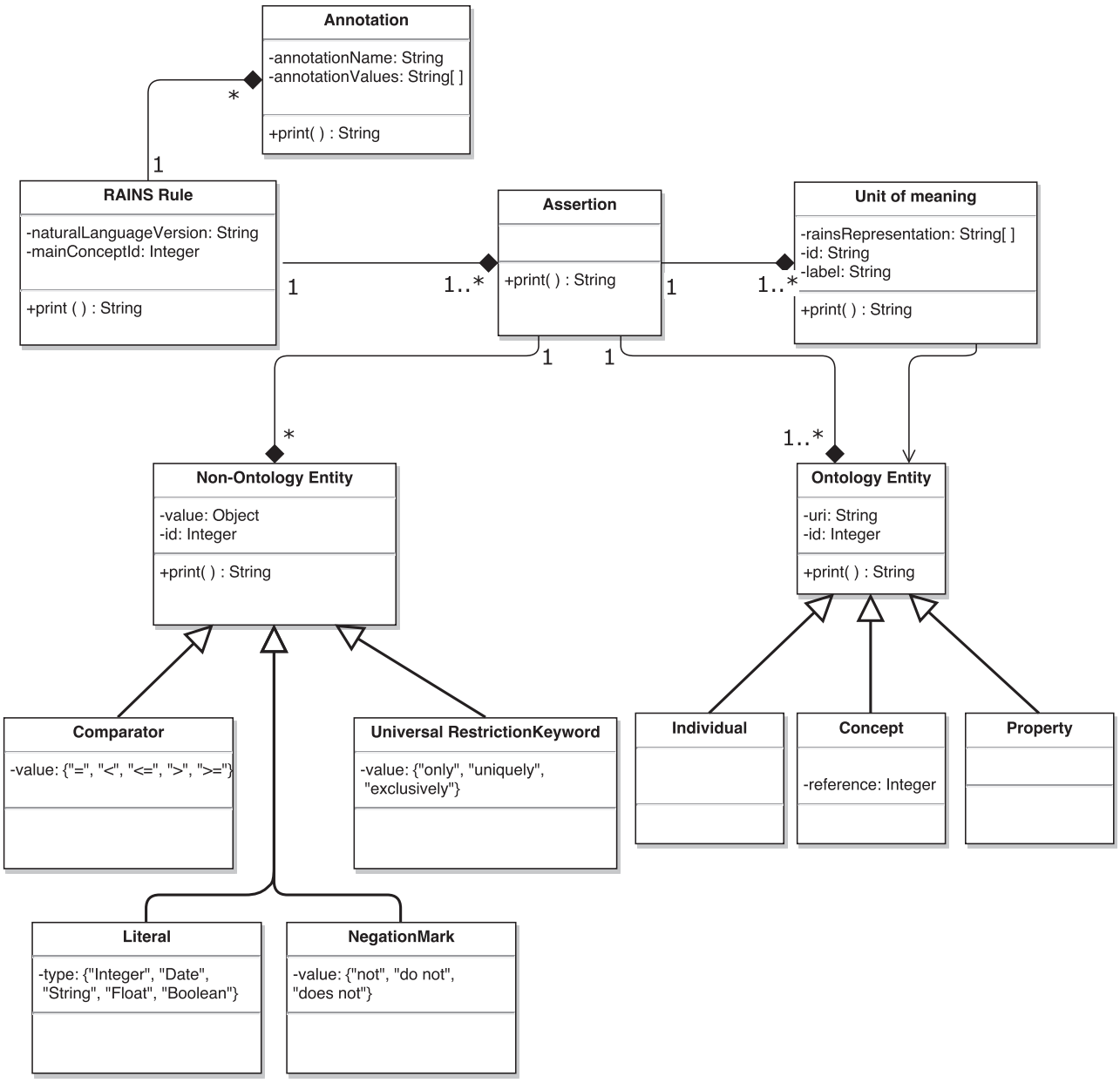


FIGURE 6.1 – Diagramme de Classes UML décrivant la représentation d'une règle RAINS

Afin de modéliser les différentes versions RAINS d'une exigence à l'aide de ce schéma, nous *"**normalisons**"* ces règles RAINS. C'est cette opération qui permet d'écarter les différences syntaxiques entre des versions RAINS d'une exigence. Cette normalisation consiste à :

1. Mettre toutes les assertions selon la forme ordonnée correspondante, c'est-à-dire *ordered data assertion* ou *ordered object assertion*, etc. Par exemple
   `If the "rated speed" of a "Lifting platform" is "greater than" "0.15"`
   devient
   `If a "Lifting platform" has a "rated speed" that is "greater than" "0.15"`

2. Fusionner les entités pour former des *chain of object properties*. Pour illustration, bien vouloir se référer aux exemples d'introduction du langage RAINS (section 4.4 page 87)

Chacune de ces étapes doit s'accompagner ***des mises à jour des pointeurs de co-références***. Ce schéma a été retranscrit de manière processable par le biais d'une spécification XML. Pour le détail de cette spécification, voir l'annexe C. Nous nous appuyons sur l'exemple-ci après pour expliquer ce schéma :

— **Norme :** *"The rated speed of the lifting platform shall not be greater than 0,15 m/s."*
— **Propositions :**
  — Expert 1 :
    `If the "rated speed" of a "Lifting platform" is "not" "less than or equal to" "0.15" m/s`
    `Then it is "Non-compliant"2`
  — Expert 2 : `If the "rated speed" of a "Lifting platform" is "greater than" "0.15" m/s`
    `Then it is "Non-compliant"2`
— **Normalisation :** on met les assertions sous une forme ordonnée (on voie que maintenant la seule différence réside au niveau des entités de couleur en rouge)
  — `If a "Lifting platform" has a "rated speed" that is "not" "less than or equal to" "0.15" m/s`
    `Then it is "Non-compliant"1`
  — `If a "Lifting platform" has a "rated speed" that is "greater than" "0.15" m/s`
    `Then it is "Non-compliant"1`
— **Spécification XML :** les deux propositions sont fusionnées et on obtient le résultat exposé par le *listing* 6.1.

Listing 6.1 – Exemple de spécification des propositions de règles RAINS

```
<rainsRule>
 <id>1</id>
 <nlVersion>The rated speed of the lifting platform shall not be greater than 0,15 m/
     s.</nlVersion>
 <assertion>
  <ontologyEntity id="1" type="concept">
   <uri>:LiftingPlatform</uri>
  </ontologyEntity>
  <ontologyEntity id="2" type="property">
   <uri>:ratedSpeed</uri>
  </ontologyEntity>
  <nonOntoLogyEntity id="3" type="negationMark">
   <value>not</value>
  </nonOntoLogyEntity>
  <nonOntoLogyEntity id="4" type="comparator">
```

```
15    <value>less than or equal to</value>
16   </nonOntoLogyEntity>
17   <nonOntoLogyEntity id="5" type="comparator">
18    <value>greater than</value>
19   </nonOntoLogyEntity>
20   <nonOntoLogyEntity id="6" type="literal">
21    <value>0.15</value>
22   </nonOntoLogyEntity>
23   <unitOfMeaning id="LP">
24    <label>lifting platform</label>
25    <rainsRep>1</rainsRep> <!-- entity 1 = "Lifting platform"-->
26   </unitOfMeaning>
27   <unitOfMeaning id="rs">
28    <label>rated speed</label>
29    <rainsRep>2</rainsRep>
30   </unitOfMeaning>
31   <unitOfMeaning id="gt">
32    <label>greater than</label>
33    <rainsRep>3 4</rainsRep> <!-- the combination of entities 3 and 4-->
34    <rainsRep>5</rainsRep>
35   </unitOfMeaning>
36   <unitOfMeaning id="f">
37    <label>0.15</label>
38    <rainsRep>6</rainsRep>
39   </unitOfMeaning>
40   <rainsExpression>LP rs gt f</rainsExpression>
41  </assertion>
42  <mainConceptId>1</mainConceptId>
43 </rainsRule>
```

L'exemple ci-dessus nous permet de souligner le fait que ce sont les unités de sens (*units of meaning*), qui permettent de gérer la multiplicité dans le choix des entités. En effet, on peut noter que l'expression RAINS finale de l'assertion (donnée par la balise `<rainsExpression>`) qui constitue l'antécédent (voir ligne 40) est formée d'***identifiants des unités de sens***. Or chaque unité de sens peut être semi-formalisée de plusieurs façons (voir balises `<rainsRep>`). Dans cet exemple, l'expression RAINS est `LP rs gt f` avec :

- `LP` qui représente l'unité de sens derrière l'expression *"lifting platform"* (ligne 24) qui est formalisée par l'entité d'identifiant 1 (ligne 25) à savoir `:LiftingPlatform` (lignes 5-7).
- Idem en ce qui concerne l'unité de sens `rs` (ligne 27), l'expression *"rated speed"* et l'entité identifiée par 2 (ligne 29) qui est `:ratedSpeed` (ligne 9 à 11)
- Même chose pour le littéral `f` (ligne 36), de libellé *"0.15"* et l'entité non-ontologique d'identifiant 6 (ligne 38) qui est le flottant `0.15` (ligne 20 à 22)
- En revanche l'identifiant `gt` qui fait référence à l'unité de sens codée par *"greater than"* peut être représentée soit par la combinaisons des entités 3 et 4, c'est-à-

dire "not" "less than or equal to", soit par l'entité 5, c'est-à-dire "greater than".

L'expression RAINS fournie dans la spécification agit donc comme un ***patron*** (*pattern*) qui peut être instancié de manières aussi variées que la multiplicité des représentations RAINS des unités de sens.

Pour finir notons que pour le conséquent de la règle, il nous suffit de connaitre le nombre définissant la co-référence entre le concept de non-conformité et le concept principal (cf. section 5.4). Ce nombre est représenté par la valeur de la balise `<mainConceptId>`. Dans cet exemple il vaut 1 et fait référence à l'entité d'identifiant 1 qui est `:LiftingPlatform` (voir ligne 42).

Cette spécification XML s'appuie sur une spécification similaire que nous avons proposée dans [Kacfah Emani *et al.* 2015a], pour la représentation des expressions formelles de définitions de concepts en OWL dans le benchmark BEAUFORD. Nous nous appuyons sur ces travaux car, tout comme la formalisation des règles, celle des définitions peut produire plusieurs résultats ; ces derniers devant être représentés d'une manière appropriée pour faciliter la comparaison des résultats.

La spécification ci-dessus présentée, nous pouvons dès à présent définir les métriques permettant d'évaluer une approche de formalisation de règle. De même que pour la spécification, ces métriques s'inspirent des métriques du benchmark BEAUFORD.

6.3.2 Métriques

Pour définir nos métriques, nous nous appuyons sur les éléments de notations suivants, pour chaque règle R respectant la modélisation décrite précédemment :

— M_R : l'ensemble des unités de sens (*units of Meaning*) des assertions de la règle.

— A_R : l'ensemble des assertions et annotations de la règle

Pour deux règles $R1$ et $R2$, on définit les fonctions suivantes :

— $matching(M_{R1}, M_{R2})$ qui renvoie le sous-ensemble des unités de sens de M_{R1} qui sont correctement formalisées dans M_{R2}.

Pour une approche, et pour chaque règle R de l'ensemble de règles $\{R\}$ on note Ra la règle proposée par l'approche et Re la règle proposée par les experts et qui sert de référence.

Définition 10. La $\mathcal{M}$***-précision*** désigne le ratio du nombre d'unités de sens identifiées avec exactitude par l'approche par rapport au nombre total d'unités de sens identifiées par l'approche.

$$\mathcal{M}\text{-précision} = \frac{\Sigma_{\{R\}} \left(|matching(M_{Ra}, M_{Re})|\right)}{\Sigma_{\{R\}}|M_{Ra}|} \tag{6.1}$$

Définition 11. Le $\mathcal{M}$***-rappel*** désigne le ratio du nombre d'unités de sens identifiées avec exactitude par l'approche sur le nombre d'unités de sens proposées par les experts.

$$\mathcal{M}\text{-rappel} = \frac{\Sigma_{\{R\}} \left(|matching(M_{Ra}, M_{Re})|\right)}{\Sigma_{\{R\}}|M_{Re}|} \tag{6.2}$$

Définition 12. La ***$\mathcal{A}$-précision*** est identique à la $\mathcal{M}$-précision avec les assertions qui jouent le rôle des unités de sens

$$\mathcal{A}\text{-précision} = \frac{\Sigma_{\{R\}}\left(|A_{Ra} \cap A_{Re}|\right)}{\Sigma_{\{R\}}|A_{Ra}|} \tag{6.3}$$

Définition 13. De même, le ***$\mathcal{A}$-rappel*** reprend la définition du $\mathcal{M}$-rappel en l'adaptant pour les assertions.

$$\mathcal{A}\text{-rappel} = \frac{\Sigma_{\{R\}}\left(|A_{Ra} \cap A_{Re}|\right)}{\Sigma_{\{R\}}|A_{Re}|} \tag{6.4}$$

Les rappels et les précisions étant définis, on peut définir leur moyenne harmonique, appelée ***mesure F1***, pour avoir une vue unifiée de l'approche de formalisation en termes d'identification des unités de sens et d'identification des assertions.

$$\mathcal{M}\text{-F1} = \frac{2 \times \mathcal{M}\text{-précision} \times \mathcal{M}\text{-rappel}}{\mathcal{M}\text{-précision} + \mathcal{M}\text{-rappel}} \qquad \mathcal{A}\text{-F1} = \frac{2 \times \mathcal{A}\text{-précision} \times \mathcal{A}\text{-rappel}}{\mathcal{A}\text{-précision} + \mathcal{A}\text{-rappel}}$$

Définition 14. La ***confiance*** désigne le ratio du nombre d'exigences réglementaires correctement formalisées sur le nombre total d'exigences.

Exemple d'application. Nous illustrons ces métriques en les appliquant sur le corpus constitué des trois exigences que nous utilisons depuis le chapitre 4. Ces exigences ont été formalisées au chapitre 5 (section 5.5) et pour les propositions des experts pour ces exigences, se reporter à l'annexe C.

1. *"The rated speed of the lifting platform shall not be greater than 0,15 m/s"*
   Proposition de FORSA
   ```
   If "rated speed" "Lifting platform" "greater than" "0.15"
   Then it is "Non-compliant"2
   ```
   Les 4 unités de sens suggérer par les experts sont correctement identifiés. Il en va de même de l'unique assertion qui constitue l'antécédent. Par ailleurs, la règle RAINS produite par FORSA est en tout point correcte, au regard de la proposition des experts.
2. *"In buildings with public access, the platform length shall not be less than 1400 mm, to enable sufficient space for an attendant."*
   Proposition de FORSA
   ```
   Given "Lifting platform" "length" "less than" "1400"
   Then it is "Non-compliant"1
   ```
   Les 4 unités de sens identifiées par les experts sont convenablement formalisées. FORSA a su identifier sans erreur l'unique assertion de l'antécédent, mais en revanche, FORSA n'a pas su identifier l'annotation qui accompagne celle-ci. Ceci entraîne automatiquement le fait que la règle finale produite par RAINS soit (dans l'ensemble) incorrecte.

3. *"For doors with multiple panels, one of the panels must have a minimal width of 0.80 meter"*
   Pas de proposition provenant de FORSA. Ainsi, aucune des 6 unités de sens n'a été correctement identifiée. Idem pour les 2 assertions qui constituent l'antécédent de la règle RAINS, telle que proposée par les experts.

Avec ces éléments, nous obtenons les résultats suivants (les chiffres en indice indiquent le numéro de l'exigence à savoir 1, 2 ou 3) :

$$\mathcal{M}\textbf{-précision} = \frac{4_1 + 4_2 + 0_3}{4_1 + 4_2 + 0_3} = 1.0 \qquad \mathcal{M}\textbf{-rappel} = \frac{4_1 + 4_2 + 0_3}{4_1 + 4_2 + 6_3} = 0.57$$

$$\mathcal{A}\textbf{-précision} = \frac{1_1 + 1_2 + 0_3}{1_1 + 1_2 + 0_3} = 1.0 \qquad \mathcal{A}\textbf{-rappel} = \frac{1_1 + 1_2 + 0_3}{1_1 + 2_2 + 2_3} = 0.4$$

Confiance. Elle est égale à 1/3, car une seule des exigences, la première, a été correctement formalisée de bout en bout.

6.3.3 Résultats

Après avoir appliqué FORSA, sur le corpus décrit à la section 6.2.1, nous obtenons les résultats présentés dans le tableau 6.2.

Tableau 6.2 – Résultats de l'évaluation de l'approche FORSA (le symbole # veut dire ***nombre***)

$\#\mathbf{M_{R_a}}$ **correctes**	$\#\mathbf{M_{R_a}}$ **extraites**	$\#\mathbf{M_{R_e}}$	$\mathcal{M}$**-précision**	$\mathcal{M}$**-rappel**	$\mathcal{M}$**-F1**
173	203	276	0.85	0.63	**0.72**
$\#\mathbf{A_{R_a}}$ **correctes**	$\#\mathbf{A_{R_a}}$ **extraites**	$\#\mathbf{A_{R_e}}$	$\mathcal{A}$**-précision**	$\mathcal{A}$**-rappel**	$\mathcal{A}$**-F1**
52	59	77	0.88	0.66	**0.75**

Sur les 50 exigences présentes dans le corpus, 31 ont été conformes aux choix de formalisation des experts. Ceci correspond à une confiance de **0.62**.

6.3.4 Analyse des erreurs

En analysant les résultats obtenus par FORSA et en les comparant aux formalisations proposées par les experts métiers, il en ressort cinq principales sources d'erreurs. Nous les répartissons en deux catégories à savoir, les erreurs provenant des outils utilisés par FORSA et les erreurs dues aux heuristiques utilisées dans l'approche FORSA. Chacune de ces erreurs a été à l'origine de la non-identification d'un nombre d'unités de sens donné. Pour chaque type d'erreur, le nombre d'unités de sens non identifiées est consigné dans le tableau 6.3. Notons que le nombre total d'erreurs (103) est égal à $\#M_{Re} - \#M_{Ra}$-correctes (voir tableau 6.2).

Tableau 6.3 – Répartition des causes d'erreurs dans FORSA.

Erreurs dues aux outils		**Erreurs propres à FORSA**		
CSD-IE	*Illinois Chunker*	Entités non-onto.	Entités onto.	Configurations
12 (11.7%)	17 (16.5%)	27 (26.2%)	31 (30.1%)	16 (15.5%)

1. Erreurs dues aux outils utilisés par FORSA

Extraction d'informations. FORSA a recours à l'extraction d'informations pour identifier les assertions des exigences (section 5.1. Ces dernières suggèrent, heuristiquement, à FORSA les assertions semi-formelles devant constituer la règle RAINS recherchée. Ainsi, une identification erronée des assertions-LN est fortement susceptible d'entraîner des erreurs, voire une interruption du processus de formalisation. L'outil d'extraction d'informations utilisé dans FORSA est CSD-IE [Bast & Haussmann 2013]. Tout comme FORSA, CSD-IE utilise des outils, notamment le Stanford parser pour les tâches de base du TALN (*tokenization*, étiquetage morpho-syntaxique, détermination de l'arbre des constituants, etc.), et disposent de ses heuristiques propres. Une erreur dans l'étiquettage morpho-syntaxique des termes de l'exigence peut avoir une incidence sur l'arbre des constituants de la phrase. Or, c'est sur ce dernier que repose principalement CSD-IE. Une solution dans ce cas, serait d'entraîner (au sens *machine learning*) le Stanford parser sur des corpus spécifiques à notre domaine d'application et prendre ce changement en compte dans l'implémentation de CSD-IE. En ce qui concerne CSD-IE proprement dit, une analyse de ses sources d'erreur a été publiée dans [Kacfah Emani *et al.* 2014]. Il en ressort que les erreurs proviennent notamment de la séparation non-souhaitée des mots des syntagmes (*multi-word expressions*) et de la mauvaise identification des items des énumérations. La gestion des énumérations restent un problème complexe à gérer en TALN. Quant à la gestion des syntagmes, nous avons proposé une solution dans [Kacfah Emani *et al.* 2014]. Cette solution repose principalement sur la ***compression*** des syntagmes avant l'extraction et leur décompression après l'extraction d'informations proprement dite. Cependant, elle n'est opérée que pour les syntagmes représentant des mots du vocabulaire de domaine. De plus, ces derniers doivent être présents dans les exigences sous une forme identique à celle dans laquelle ils apparaissent dans le vocabulaire.

Identification des syntagmes. FORSA se sert de la détection des syntagmes pour identifier les potentielles unités de sens au sein d'une assertion (voir section 5.2.1). Si des syntagmes sont erronés, cela peut conduire à une mauvaise identification des entités. FORSA utilise l'*Illinois chunker* [Punyakanok & Roth 2001] pour effectuer cette tâche. Or, ce dernier utilise des classificateurs (en anglais *classifiers*) qui ont été entraînés (au sens *machine learning*) sur des corpus précis. Comme pour l'extraction d'information, un en-

traînement de l'*Illinois chunker*, sur un corpus spécifique à notre domaine d'application devrait améliorer les performances de la détection des syntagmes.

2. Erreurs dues aux heuristiques utilisées dans l'approche FORSA

Identification des entités non-ontologiques. L'identification des entités ontologiques est une des étapes essentielles de l'approche FORSA. Elle intervient dans la transformation d'une assertion encore en LN en une assertion RAINS (voir section 5.2.2). Cette tâche permet de définir l'allure que doit avoir la future assertion RAINS en passant les configurations RAINS par un flitre, pour ne conserver que quelques configurations candidates. Une erreur au cours de cette étape peut non seulement induire des erreurs dans l'assertion RAINS finale, voire interrompre le processus de formalisation (voir exemple 3 à la section 5.5 en page 156). Jusqu'ici nous détectons les types d'entités hors ontologies en se basant les formes les plus courantes dans lesquelles on peut les retrouver (voir listing 4.1). Par exemple, les termes suggérant un marqueur de négation sont *not*, *no*, *do not*, *have no*, etc. Or, il peut arriver que la négation soit encodée par des expressions telles que *without* (***sans***), *lack* (***manquer de***), *fail* (***manquer***, ***échouer***), etc. On peut aussi prendre l'exemple des nombres qui ne sont pas toujours des littéraux. On l'a vu dans l'exemple 3 à la page 156 ou le terme *one* (***un***) désigne un article plutôt qu'un littéral. Les heuristiques employées pour la détection des entités hors ontologies dans FORSA doivent donc être améliorées.

Identification des entités ontologiques - Influences des seuils. Au cours de l'étape d'identification des entités ontologiques (section 5.2.4, pages 124 et suivantes), FORSA utilise deux valeurs seuils pour restreindre l'espace de recherche des entités candidates. Pour un terme t en LN donné, FORSA considère que seuls les n premiers (ranger par ordre décroissant de similarité entre chaîne de caractères) sont pertinents. De plus, parmi ces n candidats, uniquement les n' qui ont une mesure de similarité supérieure à un seuil σ, sont considérés comme étant significatifs. En ayant recours aux seuls n et σ, FORSA peut éliminer l'entité ontologique la plus à même de représenter formellement le terme t. Or une mauvaise résolution de ce terme peut entrainer une mauvaise configuration ou une configuration non sémantiquement valide (au regard du graphe décrit par l'ontologie du domaine). Pour l'évaluation de FORSA, nous avons utilisé les valeurs 3 pour n et 0.4 pour σ. Ces valeurs découlent des expériences d'alignement d'entités, que nous avons menées pour la formalisation des définitions [Kacfah Emani *et al.* 2015b; 2016b]. Cependant, ces valeurs de seuils ont été obtenues en travaillant avec des ontologies différentes d'IfcOWL. Un travail d'adaptation de ces seuils, notamment σ devrait conduire à des valeurs différentes.

Nombre restreint des configurations. Chaque assertion RAINS dispose d'une configuration précise qui reflète la position et le type d'entités RAINS contenues dans cette

assertion. La liste complète des assertions RAINS est déterminée par la syntaxe formelle de RAINS (voir *listing* 4.1). Nous avons vu lors de la présentation de FORSA que ces configurations nous servent de calque pour recopier convenablement les entités ontologiques et non-ontologiques identifiées au cours de la formalisation d'une assertion-LN. Ce travail de calque joué par les configurations RAINS est d'autant plus efficace que les entités présentes dans les assertions-LN sont du même type et disposés dans le même ordre que les types et les ordres exigés par la syntaxe de RAINS (cf. algorithme **??**). Or, le nombre et la diversité des configurations d'assertions proposées par RAINS restent limités par rapport aux configurations qu'offrent le langage naturel. Ainsi, lorsque la configuration d'une assertion-LN ne se retrouve pas dans le champ des configurations offert par RAINS, FORSA aligne la configuration de l'assertion en LN sur la configuration RAINS la plus proche. Cette opération n'est pas exempte d'erreurs. En plus de l'amélioration de la procédure de recherche de la configuration RAINS adéquate correspondant à une configuration en LN inconnue de RAINS, une solution serait d'enrichir la syntaxe de RAINS de nouvelles expressions et donc d'étendre le champ des configurations RAINS.

6.4 Conclusion du chapitre

Les principales contributions de notre travail de thèse sont contenues dans le langage RAINS et dans l'approche FORSA. Nous avons consacré ce chapitre à l'évaluation de ce ces deux objets d'étude.

Le matériel pour conduire ces évaluations se compose d'un corpus d'exigences réglementaires à formaliser et d'une ontologie de domaine. Pour constituer le corpus d'évaluation, nous avons extrait, complètement au hasard, de la norme [NF EN 81-41 2011], 50 exigences ayant le profil adéquat. Ce profil impose de respecter la typologie des normes rentrant dans le cadre de notre travail (voir section 3.1.5) à savoir que ces exigences doivent être des phrases contenant des prescriptions strictes, sans références croisées ou implicites et complètement interprétables. Ensuite, à partir du glossaire de cette norme nous avons pu enrichir l'ontologie IfcOWL originale à l'aide de termes propres au domaine ciblé par la norme ; à savoir le domaine des élévateurs.

L'évaluation du langage RAINS a consisté à familiariser les experts avec ce dernier, en les demandant de transformer les 50 exigences de leur forme originale en règles RAINS de détection de non-conformité (voir section 6.2). Nous avons ainsi pu recueillir leurs impressions sur l'utilisabilité de RAINS. Aussi, nous avons analysé les sources d'erreurs commises par les experts. Il en est ressorti que RAINS est un langage qui permet aux experts de manipuler les règles formelles sans avoir besoin d'assistance. De plus, mettre une exigence complexe sous une forme RAINS facilite l'échange entre experts

sur un même corpus. En revanche, le mécanisme de spécification des co-références dans RAINS n'est pas naturel et est une importante source d'erreurs. Enfin, le schéma IFC étant vaste, il serait souhaitable de disposer d'un éditeur de règles RAINS qui facilite et garantisse l'utilisation d'entités IFC valides dans les règles. Enfin, notons que l'évaluation de RAINS a permis de constituer un corpus de référence, de 50 exigences disposant d'une version RAINS. Cette base de référence nous a servi pour l'évaluation de FORSA. L'évaluation de l'approche FORSA peut se résumer en une comparaison entre les résultats fournis par FORSA et les résultats proposés par les experts métiers (voir section 6.3). Sachant que pour une exigence donnée, on peut aboutir à plusieurs règles RAINS (différentes de par la syntaxe et/ou de par le choix des entités), nous avons proposé une modélisation qui permettent d'avoir une vue adéquate sur le résultat de la formalisation afin de permettre cette comparaison. Ladite comparaison s'appuie sur cinq (5) principales métriques. La $\mathcal{M}$***-précision*** et le $\mathcal{M}$***-rappel*** qui sont des métriques qui jaugent la capacité de l'approche à formaliser convenablement les différentes unités de sens contenues dans les exigences. La $\mathcal{A}$***-précision*** et le $\mathcal{A}$***-rappel***, quant à elles, évaluent la faculté de l'approche à formaliser avec justesse les différentes assertions qui composent les exigences. Enfin, on distingue la mesure de ***confiance***, qui indique le ratio d'exigences dont la formalisation est en tout point irréprochable. FORSA obtient une moyenne de 0.72 pour l'identification des unités de sens, de 0.75 pour sa capacité à formaliser les assertions et une confiance globale de 0.62. En effet, 31 des 50 exigences du corpus ont été convenablement transformées en règles RAINS. En outre, une analyse des erreurs de FORSA fait ressortir deux principales origines. En effet, les erreurs sont dues soient à des résultats inexacts fournis par les outils supports (CSD-IE pour l'extraction d'information et l'*Illinois chunker* pour la détection des syntagmes), soient à des résultats indésirables obtenus après l'application des heuristiques propres à FORSA. Nous avons analysé chacune de ses sources d'erreurs et avons apporté des propositions de solutions.

Chapitre 7

Conclusion et Perspectives

Contents

Le monde de la Construction est en train d'amorcer une transition numérique. L'une des missions de ce plan de transition est de "démontrer l'efficacité du numérique à toutes les étapes de l'acte de construction, puis en phase d'exploitation, de maintenance, d'entretien et de rénovation" [1]. Cette efficacité s'observe au niveau de la modélisation et du partage des données, en matière d'optimisation et d'appréciation des coûts, ou encore sur l'amélioration de la qualité et de la sécurité des bâtiments. Ces deux derniers aspects concernent principalement le respect des réglementations en matière de construction. Pour que les outils de manipulation de projets de construction, sous une forme numérique, puissent s'assurer du respect des normes, il est absolument nécessaire que ces dernières soient elles aussi sous une forme numérique et processable. Or transformer tout le corpus technico-réglementaire sous une forme numérique exploitable par l'ordinateur pour les besoins d'un contrôle de conformité est une opération couteuse. En effet, elle exige que des personnes ayant une excellente compréhension des normes se consacrent à la formalisation de l'ensemble du corpus technico-réglementaire de la Construction (près de 4511 documents couvrant environ 76000 pages). Par ailleurs, ces spécialistes n'ont pas nécessairement des compétences en représentation des connaissances. Cet état de fait a conduit les experts de la Construction à collaborer avec des experts en représentation de connaissances pour la constitution de corpus de normes sous une forme processable.

Le but de cette thèse était de réduire le temps passé par ces experts dans cet exercice de formalisation de textes réglementaires. Pour cela nous avons effectué des propositions que nous pouvons regrouper à travers deux principales contributions. La première est un nouveau langage naturel contrôlé, appelé RAINS, pour l'écriture de règles pour le contrôle de conformité et la seconde est un service implémentant FORSA, une nouvelle approche automatique de formalisation de règles. Nous présentons un résumé de RAINS et FORSA et nous évoquons les perspectives de nos travaux de thèse.

7.1 Résumé des contributions

Notre état de l'art sur la constitution de corpus de règles formelles a montré que jusqu'à présent, cette tâche nécessitait la présence d'un expert métier et d'un expert en représentation des connaissances. Pour éviter le recours à ce second expert, nous avons fait le choix d'un compromis consistant à doter l'expert métier d'un formalisme intermédiaire pour l'expression des règles. Afin de rester dans un univers de représentation des connaissances proche des acquis de l'expert, nous proposons à celui-ci l'utilisation d'un langage naturel contrôlé. Nous avons baptisé ce langage RAINS. Pour définitive-

1. http ://www.batiment-numerique.fr/notre-mission/presentation.htm

ment renoncer à l'assistance d'un expert en représentation de connaissances, RAINS doit être un langage dont la transformation en expressions formelles est automatique et dépourvue d'heuristiques et de suppositions.

Le langage RAINS a donc été conçu de sorte à avoir une précision maximale au sens de l'échelle PENS (Précision, Expressivité, Naturalité et Simplicité) proposée par Kuhn [2014]. Pour cela, RAINS dispose d'une syntaxe formelle et les phrases RAINS peuvent être automatiquement transformées en requête SPARQL. La syntaxe de RAINS encadre l'écriture des règles RAINS qui ne peuvent plus être écrites dans un style aussi libre que le permet le langage naturel. Par exemple, RAINS délimite les termes utiles (entités RAINS) par des guillemets, permet de greffer des métadonnées aux règles, impose que les phrases soient explicitement sous la forme "si... alors", fait appel à des nombres pour signaler des co-références, etc. Notons que les entités RAINS qui forment le vocabulaire de ce langage, sont composées de termes du domaine (issus de l'ontologie IfcOWL enrichie - voir section 6.1 -, dans notre cas d'application), et de mots réservés tels que les marques de négations, les fonctions de comparaison, des marques de quantification universelle et des littéraux. De part sa syntaxe formelle, l'expressivité de RAINS et sa naturalité sont en deçà de celles du langage naturel. Toutefois, RAINS permet d'utiliser librement des expressions en langage naturel pour connecter les entités RAINS et ainsi faciliter l'écriture, la lecture et la compréhension des phrases RAINS. Nous avons vu que, le couplage de cette caractéristique à la délimitation des entités RAINS par des guillemets facilite l'écriture de règles multilingues et sémantiquement équivalentes. Quant à l'aspect simplicité qui concerne la description de la syntaxe et de la sémantique du langage, RAINS possède une longue description (plus de 10 pages). En résumé, RAINS est un langage de coordonnées (P^5, E^3, N^3, S^3) dans l'espace PENS.

RAINS permet donc à l'expert métier, garant de la connaissance et du respect des normes, d'être le seul opérateur dans la manipulation du corpus technico-réglementaire. Toutefois, réécrire manuellement toutes les exigences du corpus technico-réglementaire en règle RAINS demeure une activité qui requiert beaucoup de temps. Pour cette raison, nous avons proposé d'assister les experts dans la mise sur pied de corpus de règles formels, par l'utilisation d'un service implémentant une approche de formalisation automatique de règles. Pour cela nous avons proposé une approche dénommée FORSA et dont le but est de transformer une exigence de sa forme originale en une exigence RAINS. Le résultat de cette transformation doit être validé par l'expert métier. Rappelons que les normes traitées à travers FORSA doivent répondre à un certain nombre de critères : ces normes doivent être des phrases décrivant des règles strictes (obligations ou interdictions) et doivent être complètement interprétables. Une évaluation de FORSA sur un corpus de 50 exigences tirées d'une norme existante (la norme [NF EN

81-41 2011] relative aux règles de sécurité pour la construction et l'installation des plateformes élévatrices) permet d'obtenir la formalisation correcte de 31 exigences, soit 62%.

7.2 Perspectives

Dans l'objectif d'améliorer nos contributions nous proposons différentes perspectives.

L'extension de la syntaxe du langage RAINS. Lorsque nous avons présenté l'analyse des erreurs de l'approche FORSA, nous avons mentionné que 15.5% des échecs de FORSA étaient dus aux limites de la syntaxe du langage RAINS. FORSA ayant pour but de transformer des phrases en LN en des assertions RAINS, si la phrase en langage naturel possède une structure éloignée de celle des assertions RAINS, le risque d'erreur de la part de FORSA est élevé. Une extension du catalogue de structures que peuvent avoir les assertions RAINS augmenterait non seulement la naturalité de RAINS mais permettra en plus d'améliorer les performances d'approches de formalisation automatique telles que FORSA. Un tout autre aspect de l'enrichissement de la syntaxe de RAINS sera la prise en compte des ***formules***. En effet, dans les règles métiers on retrouve parfois explicitement des formules ou encore des termes (par exemple ***surface utile***, ***volume***) qui font référence à des valeurs devant être obtenues par un calcul. La syntaxe actuelle de RAINS n'intègre pas la déclaration de formules. Ainsi, une extension de la syntaxe par un mécanisme de déclaration de formules, tout en ayant à l'idée la non-altération de la naturalité de RAINS, est souhaitable.

Recours à l'apprentissage supervisé pour améliorer des résultats de TALN. Toujours en ce qui concerne l'étude des sources d'erreurs dans les résultats produits par FORSA, nous avons noté un impact des outils utilisés pour des opérations de TALN. Nous avons mentionné que l'identification des assertions atomiques et celle des syntagmes étaient à l'origine de plus de 28% d'erreurs dans les résultats de FORSA. Nous avons aussi souligné que les outils utilisés pour ces tâches s'appuyaient sur des classificateurs entraînés sur des corpus annotés généraux (c'est-à-dire non-spécifiques à un domaine de connaissances précis). Nous pensons que la mise sur pied des corpus annotés dédiés à notre domaine d'application, et l'entraînement des classificateurs sur ces corpus, pourraient améliorer les performances de l'extraction d'informations et aussi de la détection des syntagmes et donc avoir une influence positive sur les résultats de FORSA.

Taille et diversification du corpus d'évaluation de FORSA. L'évaluation de l'approche FORSA porte actuellement sur un corpus de 50 exigences réglementaires, toutes issues

du domaine de la normalisation des élévateurs. Il serait intéressant d'élargir et de diversifié ce corpus afin de mieux apprécier l'approche FORSA. Cette diversification pourrait se faire suivant l'axe des sous-domaines du secteur de la Construction (sécurité incendie, isolation thermique, etc.) ou en s'attardant sur des domaines différents (usinage de pièces mécaniques, gestion des avantages fidélités pour les compagnies de transport, etc.). Rappelons que bien que la syntaxe du langage RAINS ait été inspirée par la non-naturalité des termes du schéma IFC, cette syntaxe ne suppose pas que les règles RAINS soient écrites avec des termes des IFC, mais avec des termes du domaine d'application. L'ontologie de référence peut donc être IfcOWL ou toute autre ontologie.

Extension de la portée de FORSA. L'approche FORSA a été conçue pour manipuler des exigences encapsulée dans l'unité linguistique qu'est la phrase. Or, comme nous l'avons souligné en présentant le corpus technico-réglementaire de la Construction, il existe des règles exprimées dans un champ plus large qu'une phrase (c'est-à-dire plusieurs phrases ou des listes, ou un paragraphe, tableaux). Il serait intéressant de se pencher sur une adaptation de FORSA à cette pluralité d'expression de contraintes.
Nous clôturons cette série de perspectives par quelques recommandations. Au cours de notre travail, nous manipulons des normes issues de textes réglementaires existants. Or la transition numérique en cours dans le monde de la Construction ne laisse pas de doute sur la nécessité de disposer d'une version processable de chaque norme dans le futur. Pour les exigences qui seront écrites dans l'avenir, un effort pourrait être fait afin que la transformation (semi-) automatique de ces normes en expressions formelles soit facilité. Cela passe par l'écriture de phrases courtes simples (idéalement composée d'un groupe sujet, d'un groupe verbal et de compléments) et sans mention implicite ; cela couplé à un paradigme pouvant se résumer par "une phrase pour chaque exigence et une exigence par phrase". Par ailleurs, nous avons souligné la non-naturalité du standard IFC dont la formalisation des idées et les termes sont très éloignés de ce que l'on retrouve dans langage naturel. Il serait pertinent de maintenir un vocabulaire de la Construction et des domaines connexes qui soient plus proche des termes employés dans les textes technico-réglémentaires et qui permettent une formalisation plus directe des idées.

Implémentation d'un mécanisme d'apprentissage dans FORSA. Chaque règle RAINS issues d'une formalisation à travers FORSA est validée par l'expert métier. En cas d'erreur sur le résultat produit par FORSA, la validation de l'expert donnera lieu à une série de corrections. Nous pensons qu'une version améliorée de FORSA pourrait apprendre de ces corrections. Cet apprentissage pourrait être présent de manière native dans FORSA. Autrement dit, au cours des étapes de transformations d'une règle métier

en règle RAINS, on devrait retrouver des mécanismes de choix tenant compte des corrections et des formalisations validées précédemment par des experts métiers. Le corpus de règles métiers déjà transformées en règles RAINS servirait aussi, dans ce mécanisme d'apprentissage.

Annexes

Annexe A

Étiquettes morpho-syntaxiques du Penn Treebank

Voici la liste des étiquettes morpho-syntaxiques définies dans le projet Penn Treebank. Elles sont utilisées pour rattacher aux mots d'une phrase ou d'une expression, des informations grammaticales (catégorie grammaticale, genre, nombre).

CC Conjonction de coordination

CD Nombre cardinal

DT Déterminant

EX *Existential there*

FW Mot étranger

IN Préposition ou conjonction de subordination

JJ Adjectif

JJR Adjective, comparatif

JJS Adjective, superlatif

LS Marqueur d'item de liste

MD Verbe modal

NN Nom, singulier ou masse

NNS Nom, pluriel

NNP Nom propre, singulier

NNPS Nom propre, pluriel

PDT Pré-déterminant

POS Marqueur de pronom possessif

PRP pronom personnel

PRP$ Pronom possessif

RB Adverbe

RBR Adverbe, comparatif

RBS Adverbe, superlatif

RP Particule

SYM Symbole

TO *To*

UH Interjection

VB Verbe, infinitif

VBD Verbe, participe passé

VBG Verbe, participe présent

VBN Verbe, participe passé

VBP Verbe, pas à la 3e personne du singulier au présent

VBZ Verbe, à la troisième personne du singulier au présent

WDT Déterminant commençant par *Wh*

WP Pronom commençant par *Wh*

WP$ Pronom possessif commençant par *Wh*

WRB Adverbe commençant par *Wh*

Pour plus de détails sur le Penn Treebank, consulter `https://www.cis.upenn.edu/~treebank/`.

Annexe B

Algorithmes pour la résolution de l'assertion RAINS

Au chapitre 5, nous présentons FORSA, notre approche de formalisation automatique des règles métiers. FORSA prend en entrée une règle en langage naturel et rend en sortie une version RAINS de cette règle. Cette transformation est décomposée en plusieurs grandes étapes. L'une d'elles, l'étape 2 (voir section 5.2), consiste à transformer les assertions atomiques contenues dans la règle métier en assertion RAINS. Or, pour une assertion atomique donnée, il y a plusieurs assertions RAINS candidates (voir section 5.2.5). Il est ainsi nécessaire d'exhiber l'assertion RAINS candidate à la formalisation de l'assertion atomique. Cela se fait au travers de l'algorithme B.1 ci-dessous. Pour gérer des sous-cas de ce problème de résolution de l'assertion RAINS d'une assertion atomique, l'algorithme B.1 fait appel aux algorithmes B.2 - page 191 - et B.3 - page 192 - ci-dessous. Ces algorithmes sont expliqués en détails à la section 5.2.5. Des exemples détaillés d'application de ces algorithmes sont fournis au travers des exemples d'exécution de l'approche FORSA en section 5.5.

Algorithme B.1 Résolution de l'assertion RAINS

Données : $\mathcal{L}$, LV, $L\overline{V}$, $\overline{L\overline{V}}$ % $\mathcal{L}$ est la liste restreinte des configurations RAINS
Résultat : rainsAssertion

candidateAssertions = ∅;
si *LV est non vide* **alors**
  pour ass ∈ *LV* **faire**
    c = *getConfiguration*(ass);
    configToAssertionsMap.*put*(c, ass);
    ❶ *vote*($\mathcal{L}$, c); % Un vote supplémentaire pour la configuration c dans $\mathcal{L}$
  $\mathcal{L}'$ = *highestVotes*($\mathcal{L}$);
  % $\mathcal{L}'$ contient les configurations qui ont chacune le nombre de votes maximal
  ❷ **pour c** ∈ $\mathcal{L}'$ **faire**
    assertions = configToAssertionsMap.*get*(c);
    sortByScoreDesc(assertions);
    ass = assertions.*get*(0);
    % ass est l'assertion de configuration c et de score maximal
    candidateAssertions.*add*(ass);
  ❸ *sortByScoreDesc*(candidateAssertions);
  rainsAssertion = candidateAssertions.*get*(0);
sinon
  si $L\overline{V}$ *est non vide* **alors**
    pour ass ∈ *LV* **faire**
      c = *getConfiguration*(ass); configToAssertionsMap.*put*(c, ass);
      ❹ *vote*($\mathcal{L}$, c);
    $\mathcal{L}'$ = *highestVotes*($\mathcal{L}$);
    ❺ **pour c** ∈ $\mathcal{L}'$ **faire**
      assertions = configToAssertionsMap.*get*(c);
      sortByConceptsScoreDesc(assertions);
      candidateAssertions.*add*(assertions.get(0));
    ❻ *sortByConceptsScoreDesc*(candidateAssertions);
    rainsAssertion = candidateAssertions.*get*(0);
  sinon
    ❼ **pour ass** ∈ $\overline{L\overline{V}}$ **faire**
      c = *getConfiguration*(ass); configToAssertionsMap.*put*(c, ass);
      ❽ $\mathcal{L}_c$ = *nearestConfigurations*($\mathcal{L}$, c);
      %Dans ce cas, c ∉ $\mathcal{L}$. $\mathcal{L}_c$ = configurations de $\mathcal{L}$ les plus proches de c
      pour c' ∈ $\mathcal{L}_c$ faire
        vote($\mathcal{L}$, c');
    ❾ $\mathcal{L}'$ = *highestVotes*($\mathcal{L}$);
    rainsAssertion = ***getRainsAssertion***($\mathcal{L}'$, **configToAssertionsMap**); Voir algorithme B.2
Retourner rainsAssertion;

Algorithme B.2 Assertion RAINS, ayant une configuration autorisée, et représentant au mieux un ensemble de configurations non autorisées donné

Données : $\mathcal{L}'$, configToAssertionsMap
% $\mathcal{L}'$ contient les configurations autorisées les plus proches des configurations des assertions RAINS candidates
Résultat : rainsAssertion

% 1. Configurations candidates, nearestConfigurations, les plus proches de $\mathcal{L}'$
nearestConfigurations $= \emptyset$; configToDistanceMap $= \emptyset$; minDistance $= -1$;
pour k $\in$ **configToAssertionsMap.*keySet()* faire**
    distance $= 0$;
    pour c $\in \mathcal{L}'$ **faire**
        distance $=$ distance $+$ *levenshtein*(c, k);
    configToDistanceMap.*put*(k, distance)
    si distance $<$ minDistance **alors**
        minDistance $=$ distance;
pour k $\in$ **configToDistanceMap.*keySet()* faire**
    distance $=$ configToDistanceMap.*get*(k);
    si distance $==$ minDistance **alors**
        nearestConfigurations.*add*(kd.*getConfig*())
% 2. Configuration candidate optimale, candidateConfig
si nearestConfigurations.*size()* > 1 **alors**
    temp $= \emptyset$;
    pour k $\in$ **nearestConfigurations faire**
        ① **si k.*contains("O")*** $\vee$ **k.*contains("D")*** $\vee$ **k.*contains("B")* alors**
            temp.*add*(k); % k contient au moins une propriété
    si temp.*size()* $>= 1$ **alors**
        nearestConfigurations $=$ temp;
maxScore $= 0.0$;
pour k $\in$ **configToDistanceMap.*keySet()* faire**
    score $=$ *sumOfConceptsScores*(configToAssertionsMap.*get*(k));
    score $=$ score / configToAssertionsMap.*get*(k).*size*();
    si score $>$ **maxScore alors**
        ② candidateConfig $=$ k; maxScore $=$ score;

% 3. Configuration autorisée optimale, rainsConfig
sortByDistance($\mathcal{L}'$, candidateConfig);
% $\mathcal{L}'$ est triée par ordre de proximité avec **candidateConfig**
sortByNumberOfPropertiesDesc($\mathcal{L}'$); % Pour les éléments de $\mathcal{L}'$ situés à égale distance de **candidateConfig**, on privilégie ceux ayant le plus de propriétés
assertions $=$ configToAssertionsMap.*get*(candidateConfig);
③ *sortByConceptsScoreDesc*(assertions);
④ candidateAss $=$ assertions.*get*(0);
⑤ **pour rainsConfig** $\in \mathcal{L}'$ **faire**
    rainsAssertion = *getRainsAssertion*(rainsConfig, candidateConfig, candidateAss);
    si rainsAssertion $!=$ ***null* alors**
        Retourner rainsAssertion;
⑥ **Retourner** *null*;

Algorithme B.3 Détermination d'une assertion RAINS de configuration non autorisée à partir d'une configuration RAINS autorisée

```
Données : rainsConfig, candidateConfig, candidateAss
Résultat : rainsAssertion
rainsAssertion = candidateAss;
position = 0;
unauthorised = false;
① editSequence = levenshteinEditSequence(candidateConfig, rainsConfig);
% liste ordonnée des opérations pour transformer candidateConfig en
rainsConfig lors du calcul de la distance de Levenshtein
② pour i = 1 à editSequence.length() faire
    operation = editSequence.charAt(i);
    letterCode = rainsConfig.charAt(i);
    suivant operation faire
        case operation = "Delete" % Suppression
            rainsAssertion.deleteEntityAt(i);
        case operation = "Insert" % Insertion
            si letterCode = "O" alors
                rainsAssertion.insertEntityAt("undetermined property", i);
            sinon
                unauthorised = true;
        case operation = "Substitute" % Remplacement
            si letterCode = "O" alors
                rainsAssertion.replaceEntityAt("undetermined property", i);
            sinon
                unauthorised = true;
        % Si l'operation est "None" on n'effectue pas de changement.
    si unauthorised alors
        Retourner null;
Retourner rainsAssertion;
```

Éléments de notation dans les algorithmes. Nous présentons la mise en forme (***police***) et les conventions de nommage (***minuscule***, ***majuscule***, ***singulier***, ***pluriel***, ***suffixes***) employées dans les algorithmes :

— `variable` ou `variable`**`s`** (***pluriel*** pour une liste ou un ensemble)

— *fonctions*(`paramètre1, paramètre2, ...`)

— % commentaires

— `keyToValue`**`Map`** désigne ***table de hachage*** (*hashtable*) c'est-à-dire un ensemble de couples ***(key, value)***. Chaque *key* est ***unique***. Dans nos algorithmes, nous utilisons quatre méthodes des tables de hachage : (*i*) *get*(`key`) qui retourne la valeur correspondant à `key` (*ii*) *containsKey*(`key`) qui renvoie la valeur ***vraie*** (*true*) si la table de hachage contient la clé `key` et *false* sinon, (*iii*) *put*(`key, value`) qui ajoute un couple (`key, value`) à la table et (*iv*) *keySet*() qui renvoie la liste des clés.

— `keyToValue`**`s`**`Map` qui se différencie de la précédente par le fait qu'une clé est associée à une liste de valeurs
— le symbole "*!*" dans les énoncés `si . . . alors` représente la ***négation***.

Annexe C

Spécification XML d'un corpus de règles RAINS

Une exigence peut donner lieu à plusieurs règles RAINS. Ces dernières peuvent différer de part la syntaxe ou de part le choix des entités utilisées. Pour pouvoir comparer le résultat d'une approche de formalisation avec les résultats de références, il est nécessaire de disposer d'une représentation adéquate afin d'effectuer la comparaison. Pour cela, nous proposons le schéma XSD [1] suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://localhost"
xmlns="http://localhost"
elementFormDefault="qualified">
<xs:complexType name="OntoLogyEntityType">
 <xs:sequence>
  <xs:element name="uri" type="xs:string"/>
  <xs:choice minOccurs="0" maxOccurs="1">
   <xs:element name="reference" type="xs:integer"/><!--for concepts only-->
  </xs:choice>
 </xs:sequence>
 <xs:attribute name="id" type="xs:integer"/>
 <xs:attribute name="type" type="xs:string"/> <!--type = concept|individual|property-->
</xs:complexType>

<xs:complexType name="NonOntoLogyEntityType">
 <xs:sequence>
  <xs:element name="value" type="xs:string"/>
 </xs:sequence>
 <xs:attribute name="id" type="xs:integer"/>
 <xs:attribute name="type" type="xs:string"/><!-- type = comparator|literal|negation mark|
     universal restriction keyword -->
</xs:complexType>
```

1. XML Schema Definition

```
<xs:complexType name="AnnotationType">
 <xs:sequence>
  <xs:element name="name" type="xs:string"/>
  <xs:choice minOccurs="1" maxOccurs="unbounded">
   <xs:element name="value" type="xs:string"/>
  </xs:choice>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="UnitOfMeaningType">
 <xs:sequence>
  <xs:element name="label" type="xs:string"/>
  <xs:choice minOccurs="1" maxOccurs="unbounded">
   <xs:element name="rainsRep" type="xs:string"/><!-- The list of ids of rains entities that
       could be used to represent this meaning-->
  </xs:choice>
 </xs:sequence>
 <xs:attribute name="id" type="xs:string"/>
</xs:complexType>

<xs:complexType name="AssertionType">
 <xs:sequence>
  <xs:choice minOccurs="1" maxOccurs="unbounded">
   <xs:element name="ontologyEntity" type="OntoLogyEntityType"/>
  </xs:choice>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="nonOntoLogyEntity" type="NonOntoLogyEntityType"/>
  </xs:choice>
  <xs:choice minOccurs="1" maxOccurs="unbounded">
   <xs:element name="unitOfMeaning" type="UnitOfMeaningType"/>
  </xs:choice>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="implicitUnitOfMeaning" type="UnitOfMeaningType"/> <!-- Unit of meaning
       which is not explicitly mentioned in the NL requirement-->
  </xs:choice>
  <xs:element name="rainsExpression" type="xs:string"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="RAINSRuleType">
 <xs:sequence>
  <xs:element name="id" type="xs:integer"/>
  <xs:element name="nlVersion" type="xs:string"/>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="annotation" type="AnnotationType"/>
  </xs:choice>
  <xs:choice minOccurs="1" maxOccurs="unbounded">
   <xs:element name="assertion" type="AssertionType"/>
  </xs:choice>
  <xs:element name="mainConceptId" type="xs:integer"/>
 </xs:sequence>
```

```
</xs:complexType>

<xs:element name="RAINScorpus">
 <xs:complexType>
  <xs:sequence>
   <xs:choice minOccurs="1" maxOccurs="unbounded">
    <xs:element name="rainsRule" type="RAINSRuleType"/>
   </xs:choice>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Lorsque nous représentons les règles RAINS présentées en exemple au chapitre 4 en section 4.4, suivante ce schéma de représentation, nous obtenons le résultat exposé dans le listing ci-dessous.

Listing C.1 – Exemples de spécification des propositions de règles RAINS

```
<?xml version="1.0" encoding="UTF-8"?>
<RAINScorpus xmlns="http://localhost"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://localhost RAINSRule.xsd">

<rainsRule>
 <id>1</id>
 <nlVersion>The rated speed of the lifting platform shall not be greater than 0,15 m/s.</
     nlVersion>
 <assertion>
  <ontologyEntity id="1" type="concept">
   <uri>:LiftingPlatform</uri>
  </ontologyEntity>
  <ontologyEntity id="2" type="property">
   <uri>:ratedSpeed</uri>
  </ontologyEntity>
  <nonOntoLogyEntity id="3" type="negationMark">
   <value>not</value>
  </nonOntoLogyEntity>
  <nonOntoLogyEntity id="4" type="comparator">
   <value>less than or equal to</value>
  </nonOntoLogyEntity>
  <nonOntoLogyEntity id="5" type="comparator">
   <value>greater than</value>
  </nonOntoLogyEntity>
  <nonOntoLogyEntity id="6" type="literal">
   <value>0.15</value>
  </nonOntoLogyEntity>
  <unitOfMeaning id="LP">
   <label>lifting platform</label>
   <rainsRep>1</rainsRep> <!-- entity 1 = "Lifting platform"-->
  </unitOfMeaning>
  <unitOfMeaning id="rs">
```

```
33    <label>rated speed</label>
34    <rainsRep>2</rainsRep>
35   </unitOfMeaning>
36   <unitOfMeaning id="gt">
37    <label>greater than</label>
38    <rainsRep>3 4</rainsRep> <!-- the combination of entities 3 and 4-->
39    <rainsRep>5</rainsRep>
40   </unitOfMeaning>
41   <unitOfMeaning id="f">
42    <label>0.15</label>
43    <rainsRep>6</rainsRep>
44   </unitOfMeaning>
45   <rainsExpression>LP rs gt f</rainsExpression>
46  </assertion>
47  <mainConceptId>1</mainConceptId>
48 </rainsRule>
49
50 <rainsRule>
51  <id>2</id>
52  <nlVersion>In buildings with public access, the platform length shall not be less than 1400 mm
        , to enable sufficient space for an attendant.</nlVersion>
53  <annotation>
54   <name>building use</name>
55   <value>public building</value>
56  </annotation>
57  <assertion>
58   <ontologyEntity id="1" type="concept">
59    <uri>:LiftingPlatform</uri>
60   </ontologyEntity>
61   <ontologyEntity id="2" type="property">
62    <uri>:length</uri>
63   </ontologyEntity>
64   <nonOntoLogyEntity id="3" type="negationMark">
65    <value>not</value>
66   </nonOntoLogyEntity>
67   <nonOntoLogyEntity id="4" type="comparator">
68    <value>greater than or equal to</value>
69   </nonOntoLogyEntity>
70   <nonOntoLogyEntity id="5" type="comparator">
71    <value>less than</value>
72   </nonOntoLogyEntity>
73   <nonOntoLogyEntity id="6" type="literal">
74    <value>1400</value>
75   </nonOntoLogyEntity>
76   <unitOfMeaning id="LP">
77    <label>platform</label>
78    <rainsRep>1</rainsRep>
79   </unitOfMeaning>
80   <unitOfMeaning id="ln">
81    <label>length</label>
82    <rainsRep>2</rainsRep>
```

```
  </unitOfMeaning>
  <unitOfMeaning id="lt">
   <label>less than</label>
   <rainsRep>3 4</rainsRep>
   <rainsRep>5</rainsRep>
  </unitOfMeaning>
  <unitOfMeaning id="i">
   <label>1400</label>
   <rainsRep>6</rainsRep>
  </unitOfMeaning>
  <rainsExpression>LP ln lt i</rainsExpression>
 </assertion>
 <mainConceptId>1</mainConceptId>
</rainsRule>

<rainsRule>
 <id>3</id>
 <nlVersion>For doors with multiple panels, one of the panels must have a minimal width of 0.80
      meter.</nlVersion>
 <assertion>
  <ontologyEntity id="1" type="concept">
   <uri>:IfcDoor</uri>
  </ontologyEntity>
  <ontologyEntity id="2" type="property">
   <uri>:isTypedBy</uri>
  </ontologyEntity>
  <ontologyEntity id="3" type="property">
   <uri>:relatingType</uri>
  </ontologyEntity>
   <ontologyEntity id="4" type="concept">
   <uri>:IfcDoorType</uri>
  </ontologyEntity>
  <unitOfMeaning id="Dr">
   <label>doors</label>
   <rainsRep>1</rainsRep>
  </unitOfMeaning>
  <implicitUnitOfMeaning id="type">
   <label>type of Ifc Door</label>
   <rainsRep>2.3</rainsRep> <!-- The chain of properties 2 and 3 -->
  </implicitUnitOfMeaning>
  <implicitUnitOfMeaning id="DrType">
   <label>IfcDoorType</label>
   <rainsRep>4</rainsRep>
  </implicitUnitOfMeaning>
  <rainsExpression>Dr type DrType</rainsExpression>
 </assertion>
 <assertion>
  <ontologyEntity id="5" type="concept">
   <uri>:IfcDoorType</uri>
   <reference>4</reference>
  </ontologyEntity>
```

```
133  <ontologyEntity id="6" type="property">
134   <uri>:operationType</uri>
135  </ontologyEntity>
136  <ontologyEntity id="7" type="individual">
137   <uri>:DOUBLE_DOOR_FOLDING</uri>
138  </ontologyEntity>
139  <unitOfMeaning id="DoubleDr">
140   <label>multiple panels</label>
141   <rainsRep>7</rainsRep>
142  </unitOfMeaning>
143  <implicitUnitOfMeaning id="DrType">
144   <label>IfcDoorType</label>
145   <rainsRep>5</rainsRep>
146  </implicitUnitOfMeaning>
147  <implicitUnitOfMeaning id="op">
148   <label>operation of a door</label>
149   <rainsRep>6</rainsRep>
150  </implicitUnitOfMeaning>
151  <rainsExpression>DrType op DoubleDr</rainsExpression>
152 </assertion>
153 <assertion>
154  <ontologyEntity id="8" type="concept">
155   <uri>:IfcDoorType</uri>
156   <reference>4</reference>
157  </ontologyEntity>
158  <ontologyEntity id="9" type="property">
159   <uri>:hasPropertySets</uri>
160  </ontologyEntity>
161  <ontologyEntity id="10" type="property">
162   <uri>:panelWidth</uri>
163  </ontologyEntity>
164  <nonOntoLogyEntity id="11" type="negationMark">
165   <value>not</value>
166  </nonOntoLogyEntity>
167  <nonOntoLogyEntity id="12" type="comparator">
168   <value>greater than or equal to</value>
169  </nonOntoLogyEntity>
170  <nonOntoLogyEntity id="13" type="literal">
171   <value>800</value>
172  </nonOntoLogyEntity>
173  <unitOfMeaning id="width">
174   <label>have a width</label>
175   <rainsRep>9.10</rainsRep>
176  </unitOfMeaning>
177  <unitOfMeaning id="min">
178   <label>minimal</label>
179   <rainsRep>11 12</rainsRep>
180  </unitOfMeaning>
181  <unitOfMeaning id="i">
182   <label>800</label>
183   <rainsRep>13</rainsRep>
```

```
184   </unitOfMeaning>
185   <implicitUnitOfMeaning id="DrType">
186    <label>IfcDoorType</label>
187    <rainsRep>8</rainsRep>
188   </implicitUnitOfMeaning>
189   <rainsExpression>DrType width min i</rainsExpression>
190  </assertion>
191  <mainConceptId>1</mainConceptId>
192 </rainsRule>
193 </RAINScorpus>
```

Publications

Revues Internationales avec comité de lecture

— Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, and Parisa Ghodous. *BEAUFORD : A Benchmark for Evaluation of Formalisation of Definitions in OWL*. ***Open Journal Of Semantic Web***, 2 (1) :3–14, 2015a

— Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fi'es, and Parisa Ghodous. *Improving open information extraction for semantic web tasks*. ***Transactions on Computational Collective Intelligence***, pages 1–21, 2015c. doi : 10.1007/978-3-662-49521-6 6

Conférences internationales avec comité de lecture

— Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, Alain Zarli, and Parisa Ghodous. *An Approach for Automatic Formalization of Business Rules*. ***In CIB W78 Conference 2016***, pages 1–11. CIB W78, 2016b

— Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fies, Parisa Ghodous, and Marc Bourdeau. *Automated Semantic Enrichment of Ontologies in the Construction Domain*. ***In Jakob Beetz, Leon van Berlo, Timo Hartmann, Robert Amor, and Bauke de Vries, editors, CIB W78 Conference 2015***, pages 159–164, Eindhoven, The Netherlands, 2015b. CIB W78

— Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, and Parisa Ghodous. *Improving open information extraction using domain knowledge*. ***In Surfacing the Deep and the Social Web (SDSW), colocated with The 13th International Semantic Web Conference (ISWC 2014)***, volume 1310, pages 1–7. CEUR, 2014a

— Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, Parisa Ghodous, and Farzad Khosrowshahi. *Structural sentence decomposition via open information extraction*. ***In 18th International Conference Information Visualisation (IV2014)***, pages 1–6, 2014b

— Cheikh Kacfah Emani. *Automatic Detection and Semantic Formalisation of Business Rules*. ***In The Semantic Web : Trends and Challenges : 11th International Conference, ESWC - PhD Symposium*** , Anissaras, Crete, May 25-29, 2014, Proceedings, volume 8465, pages 834–844. Springer, 2014. doi : 10.1007/978-3-319-07443-6 57

Bibliographie

AFUL. Définition de l'Interopérabilité (par l'Association Francophone des Utilisateurs de Logiciels Libres). http ://definition-interoperabilite.info/, Consulté le 04/05/2016, 2016. 16

Annie I. Anton. ***Goal Identification and Refinement in the Specification of Information Systems***. PhD thesis, Georgia Institute of Technology, jun 1997. 48

Fredrik Arvidsson & Annika Flycht-Eriksson. Ontologies I. http ://www.ida.liu.se/ janma56/SemWeb/Slides/ontologies1.pdf ; Consulté le 19/05/2016, 2008. 35

Nathalie Aussenac-Gilles, Sylvie Despres, & Sylvie Szulman. The TERMINAE method and platform for ontology engineering from texts. In ***Proceedings of the 2008 Conference on Ontology Learning and Population : Bridging the Gap Between Text and Knowledge***, pages 199–223, Amsterdam, The Netherlands, 2008. IOS Press. 48

Imran S. Bajwa, Mark G. Lee, & Behzad Bordbar. SBVR business rules generation from natural language specification. In ***AAAI Spring Symposium : AI for Business Agility***, pages 2–8. AIII, 2011. 48

Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, & Oren Etzioni. Open information extraction from the web. In ***IJCAI***, volume 7, pages 2670–2676, 2007. 116

Raphael Barbau, Sylvere Krima, Sudarsan Rachuri, Anantha Narayanan, Xenia Fiorentini, Sebti Foufou, & Ram D Sriram. OntoSTEP : Enriching product model data using ontologies. ***Computer-Aided Design***, 44(6) :575–590, 2012. 39

Edward Barkmeyer & Fabian Neuhaus. RECON-A Controlled English for Business Rules. In ***RuleML***, pages 190–201, 2013. 67

Hannah Bast & Elmar Haussmann. Open Information Extraction via Contextual Sentence Decomposition. In ***7th ICSC***, pages 154–159. IEEE Computer Society, 2013. 116, 175

Tim Berners-Lee & Dan Connolly. Notation3 (N3) : A readable RDF syntax. ***W3C Team Submission***, mar 2011. https ://www.w3.org/TR/n-triples/. 43

Tim Berners-Lee, James Hendler, & Ora Lassila. The semantic web. ***Scientific american***, 284(5) :28–37, 2001. 30

C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, & M. Smith. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). ***W3C Proposed Recommendation***, dec 2012. https ://www.w3.org/TR/owl2-syntax/. 37

Alexander Boer, Rinke Hoekstra, Radboud Winkels, T Van Engers, & F Willaert. ^{META}lex : Legislation in XML. In ***Legal Knowledge and Information Systems. Jurix 2002 : The Fifteenth Annual Conference***, Frontiers in Artificial Intelligence and Applications, pages 1–10, Amsterdam, The Netherlands, 2002. IOS Press. 9, 10

Harold Boley, Said Tabet, & Gerd Wagner. Design rationale for ruleml : A markup language for semantic web rules. In ***SWWS***, volume 1, pages 381–401, 2001. 23, 24

Harold Boley, Adrian Paschke, & Omair Shafiq. RuleML 1.0 : The Overarching Specification of Web Rules. In Mike Dean, John Hall, Antonino Rotolo, & Said Tabet, editors, ***Semantic Web Rules***, volume 6403 of ***Lecture Notes in Computer Science***, pages 162–178. Springer Berlin Heidelberg, 2010. 23, 24, 42

Khalil Riad Bouzidi. ***Aide à la création et à l'exploitation de réglementations basée sur les modèles et techniques du Web sémantique***. PhD thesis, Université Nice-Sophia Antipolis, 2013. 29, 31, 45, 57

Khalil Riad Bouzidi, Bruno Fiès, Marc Bourdeau, Catherine Faron-Zucker, & Nhan Le-Thanh. An ontology for modelling and supporting the process of authoring technical assessments. In ***Proceedings of the 28th international CIB W78 conference***, 2011. 69

Khalil Riad Bouzidi, Bruno Fiès, Catherine Faron-Zucker, Alain Zarli, & Nhan Le-Thanh. Semantic web approach to ease regulation compliance checking in construction industry. ***future internet***, 4(3) :830–851, 2012. 49, 69

Dan Brcikley & R. V. Guha. RDF Schema 1.1. ***W3C Recommendation***, feb 2014. URL `https://www.w3.org/TR/rdf-schema/`. 36

Travis D. Breaux & Annie I. Anton. Analyzing goal semantics for rights, permissions, and obligations. In ***Proceedings of the 13th IEEE International Conference on Requirements Engineering***, RE '05, pages 177–188, Washington, DC, USA, 2005. IEEE Computer Society. 48, 49

Carolyn A. Brodie, Clare-Marie Karat, & John Karat. An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench. In ***Proceedings of the Second Symposium on Usable Privacy and Security***, SOUPS '06, pages 8–19, New York, NY, USA, 2006. ACM. 47

Peter Clark, William R Murray, Phil Harrison, & John Thompson. Naturalness vs. predictability : A key debate in controlled languages. In ***Controlled Natural Language***, pages 65–81. Springer, 2009. 58

Olivier Corby, Rose Dieng-Kuntz, Fabien Gandon, & Catherine Faron-Zucker. Searching the semantic Web : Approximate query processing based on ontologies. ***IEEE Intelligent Systems***, 21(1) :20–27, 2006. 45

CSTB, editor. ***Collection Guide Pratique, Les règles de construction, mieux les connaître, pour mieux les appliquer***. CSTB Éditions, 2002. xv, 14, 15

Richard Cyganiak, David Wood, & Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. ***W3C Recommendation***, feb 2014. URL `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`. 32

Juri Luca De Coi, Norbert E. Fuchs, Kaarel Kaljurand, & Tobias Kuhn. Controlled English for reasoning on the Semantic Web. In ***Semantic techniques for the web***, pages 276–308. Springer, 2009. 63

Tarcisio Mendes de Farias, Ana Roxin, & Christophe Nicolle. IfcWoD, semantically adapting IFC model relations into OWL properties. In ***Proceedings of the 32th CIB W78 International Conference***, pages 175–185, Eindhoven, The Netherlands, 2015. Jakob Beetz and Leon van Berlo and Timo Hartmann and Robert Amor and Bauke de Vries. 39

Marie-Catherine De Marneffe & Christopher D. Manning. Stanford typed dependencies manual. Technical report, Technical report, Stanford University, 2008. URL `http://nlp.stanford.edu/downloads/dependencies_manual.pdf`. 51, 137, 143

Luciano Del Corro & Rainer Gemulla. Clausie : clause-based open information extraction. In ***Proceedings of the 22nd WWW***, pages 355–366, 2013. 116

Nikos Dimaresis. A System for Modal and Deontic Defeasible Reasoning. Master's thesis, Computer Science Department, School of Sciences and Engineering, University of Crete, Heraklion, Greece, 2007. 24

Johannes Dimyadi & Robert Amor. Regulatory knowledge representation for automated compliance audit of bim-based models. In ***Proceedings of the 30th CIB W78 International Conference***, pages 68–78, 2013. 69

Johannes Dimyadi, Pieter Pauwels, Michael Spearpoint, Charles Clifton, & Robert Amor. Querying a regulatory model for compliant building design audit. In ***32rd international CIB W78 conference***, 2015. 69

Jacques Dubucs. ***Les Chemins de la logique***. Pour la science : Dossier. Éditions Belin, 2005. 22

Chuck Eastman, Charles M. Eastman, Paul Teicholz, & Rafael Sacks. ***BIM handbook : A guide to building information modeling for owners, managers, designers, engineers and contractors***. John Wiley & Sons, 2nd edition, 2011. 15

Anthony Fader, Stephen Soderland, & Oren Etzioni. Identifying relations for open information extraction. In ***Proceedings of the Conference on Empirical Methods in Natural Language Processing***, pages 1535–1545. Association for Computational Linguistics, 2011. 116

Norbert E. Fuchs, Kaarel Kaljurand, & Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In ***FLAIRS Conference***, volume 12, pages 664–669, 2006. 63

Fabien Gandon, Catherine Faron-Zucker, & Olivier Corby. ***Le web sémantique : Comment lier les données et les schémas sur le web ?*** Dunod, 2012. 32, 34, 35, 37, 38

E Garcia. Cosine similarity and term weight tutorial. ***Information retrieval intelligence***, page 5, 2006. URL `http://www.minerazzi.com/tutorials/cosine-similarity-tutorial.pdf`. 125

Vladimir Geroimenko. ***Dictionary of XML technologies and the semantic web***, volume 1. Springer Science & Business Media, 2004. 42

Thomas F. Gordon, Guido Governatori, & Antonino Rotolo. Rules and norms : Requirements for rule interchange languages in the legal domain. In ***Rule interchange and applications***, pages 282–296. Springer, 2009. 22, 23

Ian Graham. ***Business rules management and service oriented architecture : a pattern language***. John Wiley & Sons, Chichester, West Sussex, England, 2006. 47

Abdoulaye Guissé, François Lévy, & Adeline Nazarenko. Un moteur sémantique pour explorer des textes réglementaires. In Alain Mille, editor, ***Actes des 22èmes journées francophones d'Ingénierie des Connaissances***, pages 451–458. Publibook, 2011. 48

Abdoulaye Guissé, François Lévy, & Adeline Nazarenko. From regulatory texts to BRMS : how to guide the acquisition of business rules ? In ***Rules on the Web : Research and Applications***, pages 77–91. Springer, 2012. 25, 26, 27, 29

Abdoulaye Guissé. ***Une plateforme d'aide à l'acquisition et à la maintenancce des règles métier à partir de textes règlementaires***. PhD thesis, Université de Paris 13, 2013. 29

Marios Hadjieleftheriou & Divesh Srivastava. Weighted set-based string similarity. ***IEEE Data Eng. Bull.***, 33(1) :25–36, 2010. 125

Saeed Hassanpour, Martin J. O'Connor, & Amar K. Das. A framework for the automatic extraction of rules from online text. In ***Rule-Based Reasoning, Programming, and Applications***, pages 266–280. Springer, 2011. 49, 50, 51, 52, 53, 54, 55, 160

Jeff Heflin. Owl web ontology language-use cases and requirements. ***W3C Recommendation***, 10 :12, 2004. 30, 31

Stijn Heymans & Michael. Kifer. RIF Core Answer Set Programming Dialect. ***W3C Working Group Note***, dec 2009. https ://www.w3.org/TR/rif-overview/. 42

Eilif Hjelseth & Nick Nisbet. Exploring semantic based model checking. In ***Proceedings of the 2010 27th CIB W78 international conference***, volume 54, 2010. 69

Eilif Hjelseth & Nick Nisbet. Capturing normative constraints by use of the semantic mark-up (RASE) methodology. In ***Proceedings of the 28th international CIB W78 conference***, pages 1–10, 2011. 69, 70

Nam Vu Hoang & Seppo Törmä. Opening bim to the web ifc-to-rdf conversion software. http ://rym.fi/results/opening-bim-to-the-web-ifc-to-rdf-conversion-software/, 2014. 39

Rinke Hoekstra. ***Ontology RepresentationDesign Patterns and Ontologies that Make Sense***. PhD thesis, University of Amsterder - Faculty of Law, 2009. 39

Ian Horrocks & Peter F. Patel-Schneider. A proposal for an OWL rules language. In ***Proceedings of the 13th international conference on World Wide Web***, pages 723–731. ACM, 2004. 44

Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, & Mike Dean. SWRL : A Semantic Web Rule Language - Combining OWL and RuleML. ***W3C Member Submission***, may 2004. URL `https://www.w3.org/Submission/SWRL/`. 42

Willem-Olaf Huijsen. Controlled language–an introduction. In ***Proceedings of Int. Workshop on Controlled Language Applications***, volume 98, pages 1–15, 1998. 58

Cheikh Kacfah Emani. Automatic Detection and Semantic Formalisation of Business Rules. In ***The Semantic Web : Trends and Challenges : 11th International Conference, ESWC 2014 - PhD Symposium , Anissaras, Crete, Greece, May 25-29, 2014, Proceedings***, volume 8465, pages 834–844. Springer, 2014. doi : 10.1007/978-3-319-07443-6_57. 47

Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, & Parisa Ghodous. Improving open information extraction using domain knowledge. In ***Surfacing the Deep and the Social Web (SDSW), co-located with The 13th International Semantic Web Conference (ISWC 2014)***, volume 1310, pages 1–7. CEUR, 2014. 175

Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, & Parisa Ghodous. BEAUFORD : A Benchmark for Evaluation of Formalisation of Definitions in OWL. ***OJSW***, 2(1) :3–14, 2015a. 172

Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, & Parisa Ghodous. Improving open information extraction for semantic web tasks. ***Transactions on Computational Collective Intelligence***, pages 1–21, 2015b. doi : 10.1007/978-3-662-49521-6_6. 176

Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fies, Parisa Ghodous, & Marc Booourdeau. An Approach for Automatic Formalization of Business Rules. In ***CIB W78 Conference 2016***, pages 1–11. CIB W78, 2016a. 115

Cheikh Kacfah Emani, Catarina Ferreira Da Silva, Bruno Fiès, Parisa Ghodous, & Alain Zarli. NALDO : From Natural Language Definitions to OWL Expressions. http ://tinyurl.com/j6vfuj5, 2016b. 176

Kaarel Kaljurand. ***Attempto controlled English as a Semantic Web language***. Phd, University of Tartu, 2007. 63

Kaarel Kaljurand. Ace view-an ontology and rule editor based on controlled english. In ***International Semantic Web Conference (Posters & Demos)***, volume 401, 2008. 63

Kaarel Kaljurand & Tobias Kuhn. A multilingual semantic wiki based on attempto controlled english and grammatical framework. In ***The Semantic Web : Semantics and Big Data***, pages 427–441. Springer, 2013. 63

Aqueo Kamada, Guido Governatori, & Shazia Wasim Sadiq. SBVR based Business Contract and Business Rule IDE. In ***RuleML Challenge***, 2010. 66

SungKu Kang, Lalit Patil, Arvind Rangarajan, Abha Moitra, Tao Jia, Dean Robinson, & Debasish Dutta. Extraction of Manufacturing Rules From Unstructured Text Using a Semantic Framework. In ***ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference***. American Society of Mechanical Engineers, 2015. 50, 51, 53, 54, 55, 160

Michael Kifer & Harold Boley. RIF Overview (Second Edition). ***W3C Working Group Note***, feb 2013. https ://www.w3.org/TR/rif-overview/. 42

Alex Kozlenkov, Rafael Penaloza, Vivek Nigam, Loic Royer, Gihan Dawelbait, & Michael Schroeder. Prova : Rule-based java scripting for distributed web applications : A case study in bioinformatics. In ***Current Trends in Database Technology–EDBT***, pages 899–908. Springer, 2006. 43

Sylvere Krima, Raphael Barbau, Xenia Fiorentini, Rachuri Sudarsan, & Ram D Sriram. OntoSTEP : OWL-DL ontology for STEP. ***National Institue of Standards and Technology, NISTIR***, 7561, 2009. 39

Tobias Kuhn. ***Controlled English for knowledge representation***. PhD thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich, 2010. 59, 66

Tobias Kuhn. A survey and classification of controlled natural languages. ***Computational Linguistics***, 40(1) :121–170, 2014. 41, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 68, 104, 105, 110, 181

Gloria T. Lau, Kincho H. Law, & Gio Wiederhold. Legal information retrieval and application to e-rulemaking. In ***Proceedings of the 10th International Conference on Artificial Intelligence and Law***, ICAIL '05, pages 146–154, New York, NY, USA, 2005. ACM. 9, 10

Ghang Lee, Rafael Sacks, & Charles M. Eastman. Specifying parametric building object behavior (BOB) for a building information modeling system. ***Automation in construction***, 15(6) :758–776, 2006. 16

François Lévy & Adeline Nazarenko. Formalization of natural language regulations through SBVR structured english. In ***Theory, Practice, and Applications of Rules on the Web***, Lecture Notes in Computer Science, pages 19–33, Seattle, WA, USA, 2013. Springer, Heidelberg. 48

François Lévy, A. Guisse, Adeline Nazarenko, Nouha Omrane, & Sylvie Szulman. An environment for the joint management of written policies and business rules. In ***Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence***, volume 2 of ***ICTAI '10***, pages 142–149, Washington, DC, USA, 2010. IEEE Computer Society. 48

Thomas Liebich, Yoshinobu Adachi, James Forester, Juva Hyvarinen, Stefan Richter, Tim Chipman, Matthias Weise, & Jeffrey Wix. Industry Foundation Classes Release 4. ***buildingSMART International Ltd***, 2013. http ://www.buildingsmart-tech.org/ifc/IFC4/final/html/. 40

Sibel Macit, M. Emre İlal, Georg Suter, & H. Murat Günaydın. A hybrid model for building code representation based on four-level and semantic modeling approaches. In Jakob Beetz, Leon van Berlo, Timo Hartmann, Robert Amor, & Bauke de Vries, editors, ***CIB W78 Conference 2015***, pages 517–526, Eindhoven, The Netherlands, 2015. CIB W78. 70

Massimo Marchiori. Towards a people's web : Metalog. In ***Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence***, pages 320–326. IEEE Computer Society, 2004. 65

Mitchell P Marcus, Mary Ann Marcinkiewicz, & Beatrice Santorini. Building a large annotated corpus of English : The Penn Treebank. ***Computational linguistics***, 19(2) : 313–330, 1993. 119

Maussam, Michael Schmitz, Robert Bart, Stephen Soderland, & Oren Etzioni. Open language learning for information extraction. In ***Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning***, pages 523–534. Association for Computational Linguistics, 2012. 116

Jeremy C. Maxwell & Annie I. Anton. Developing production rule models to aid in acquiring requirements from legal texts. In ***Proceedings of the 2009 17th IEEE International Requirements Engineering Conference***, RE '09, pages 101–110, Washington, DC, USA, 2009. IEEE Computer Society. 48

Giuseppe M. Mazzeo & Carlo Zaniolo. CANaLI : A System for Answering Controlled Natural Language Questions on RDF Knowledge Bases. Technical Report 160004, UCLA CSD, 2016. URL `http://fmdb.cs.ucla.edu/Treports/canali_tr_160004.pdf`. 92

George A Miller. WordNet : a lexical database for English. ***Communications of the ACM***, 38(11) :39–41, 1995. 50

Teruko Mitamura & Eric Nyberg. Controlled English for knowledge-based MT : Experience with the KANT system. ***Proceedings of TMI-95***, 145 :146–147, 1995. 58

NBIMS-US^{TM}. Frequently asked questions about the NATIONAL BIM STANDARD-UNITED STATES^{TM}. https ://www.nationalbimstandard.org/faqs, Consulté le 04/05/2016, 2016. 15, 16

Saul B Needleman & Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. ***Journal of molecular biology***, 48(3) :443–453, 1970. 51

NF EN 81-41. Règles de sécurité pour la construction et l'installation des élévateurs - Élévateurs spéciaux pour le transport des personnes et des charges - Partie 41 : plateformes élévatrices verticales à l'usage des personnes à mobilité réduite. ***Association Française de Normalisation (AFNOR)***, may 2011. xv, 10, 11, 12, 13, 14, 26, 164, 165, 177, 181

Van Tien Nguyen, Christian Sallaberry, & Mauro Gaio. Mesure de la similarité entre termes et labels de concepts ontologiques. In ***Conférence en recherche d'information et applications***, pages 415–430, 2013. 125

Paul Brillant Njonko Feuto. ***Langage contrôlé pour la spécification des règles métier dans le contexte de la modélisation des systèmes d'information***. PhD thesis, Université de Franche-Comté, 2014. 58, 67

Paul Brillant Njonko Feuto, Sylviane Cardey, Peter Greenfield, & Walid El Abed. RuleCNL : A Controlled Natural Language for Business Rule Specifications. In ***Controlled Natural Language***, pages 66–77. Springer, 2014. 66

OMG. Semantics of Business Vocabulary and Business Rules (SBVR), v1.0. Technical report, Object Management Group, 2008. URL `http://www.omg.org/spec/SBVR/1.0/PDF`. 66, 67

Adrian Paschke & Harold Boley. Rules capturing events and reactivity. In ***Handbook of Research on Emerging Rule-Based Languages and Technologies : Open Solutions and Approaches***, pages 215–252. IGI Global, 2009. 41

Adrian Paschke & Harold Boley. Rule Markup Languages and Semantic Web Rule Languages. In Kuldar Taveter Adrian Giurca, Dragan Gasevic, editor, ***Handbook of Research on Emerging Rule-Based Languages and Technologies : Open Solutions and Approaches***, pages 1–24. IGI Global, 2010. 24, 41, 42, 43

Peter F. Patel-Schneider. OWL 2 Web Ontology Language New Features and Rationale (Second Edition). ***W3C Proposed Recommendation***, dec 2012. https ://www.w3.org/TR/owl2-new-features/, Consulté le 22/09/2016. 38

Pieter Pauwels & Walter Terkaj. EXPRESS to OWL for construction industry : Towards a recommendable and usable ifcOWL ontology. ***Automation in Construction***, 63 : 100–133, March 2016. 17, 39, 40, 84, 164

Pieter Pauwels, Davy Van Deursen, Jos De Roo, Tim Van Ackere, Ronald De Meyer, Rik Van de Walle, & Jan Van Campenhout. Three-dimensional information exchange over the semantic web for the domain of architecture, engineering, and construction. ***Artificial Intelligence for Engineering Design, Analysis and Manufacturing***, 25(04) : 317–332, 2011a. 49, 69

Pieter Pauwels, Davy Van Deursen, Ruben Verstraeten, Jos De Roo, Ronald De Meyer, Rik Van de Walle, & Jan Van Campenhout. A semantic rule checking environment for building performance checking. ***Automation in Construction***, 20(5) :506–518, 2011b. 38, 49, 69

Jonathan Pool. Can controlled languages scale to the web. In ***Proceedings of the 5th Int. Workshop on Controlled Language Applications***, 2006. 58

Eric Prud'hommeaux & Andy Seaborne. SPARQL Query Language for RDF. ***W3C Proposed Recommendation***, nov 2007. URL `https://www.w3.org/TR/2007/PR-rdf-sparql-query-20071112/`. 43, 45

Vasin Punyakanok & Dan Roth. The use of classifiers in sequential inference. In ***NIPS***, pages 995–1001. MIT Press, 2001. 119, 175

Robert W. Reeder, Clare-Marie Karat, John Karat, & Carolyn Brodie. Usability challenges in security and privacy policy-authoring interfaces. In ***Human-Computer Interaction – INTERACT 2007***, volume 4663 of ***Lecture Notes in Computer Science***, pages 141–155. Springer, Heidelberg, 2007. 47

Ronald G. Ross. The Business Rules Manifesto. http ://www.businessrulesgroup.org/brmanifesto.htm, nov 2003a. 76

Ronald G Ross. ***Principles of the business rule approach***. Addison-Wesley, 2003b. 66

Ronald G Ross. Rulespeak sentence forms : Specifying natural-language business rules in english. ***Business Rules Journal***, 10(4), 2009. 66

Hassan M. Satti & Robert J. Krawczyk. Issues of integrating building codes in CAD. In ***Proceedings of the 1st ASCAAD International Conference, e-Design in Architecture***, KFUPM, Dhahran, Saudi Arabia, 2004. 9

Rolf Schwitter. English as a formal specification language. In ***Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on***, pages 228–232. IEEE, 2002. 57, 58, 64

Rolf Schwitter & Marc Tilbrook. Controlled natural language meets the semantic web. In ***Proceedings of the Australasian Language Technology Workshop***, volume 2004, pages 55–62, 2004. 64

Roger S Scowen. Extended BNF-a generic base standard. Technical report, Technical report, ISO/IEC 14977. http ://www. cl. cam. ac. uk/mgk25/iso-14977. pdf, 1998. 79

Advaith Siddharthan. Syntactic simplification and text cohesion. ***Research on Language and Computation***, 4(1) :77–109, 2006. 26

Wawan Solihin & Charles Eastman. A Knowledge Representation Approach to Capturing BIM Based Rule Checking Requirements Using Conceptual Graph. In ***Proceedings of the 32th CIB W78 International Conference***, pages 686–695. Jakob Beetz and Leon van Berlo and Timo Hartmann and Robert Amor and Bauke de Vries, 2015. 69

J. F. Sowa. ***Conceptual Structures : Information Processing in Mind and Machine***. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984. ISBN 0-201-14472-7. 69

John F Sowa. Conceptual graphs for a data base interface. ***IBM Journal of Research and Development***, 20(4) :336–357, 1976. 69

Silvie Spreeuwenberg & Keri Anderson Healy. SBVR's approach to controlled natural language. In ***Proceedings of the Workshop on Controlled Natural Language***, pages 155–169. Springer, 2009. 66

Georg Henrik Von Wright. ***Norm and action***. Routledge & Kegan Paul PLC, 1963. 22

W3C. W3C Semantic Web Frequently Asked Questions. https ://www.w3.org/RDF/FAQ, Consulté le 22/09/2016, nov 2009. 31

Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damljanovic, Brian Davis, Norbert Fuchs, Stefan Hoefler, Ken Jones, Kaarel Kaljurand, Tobias Kuhn, Martin Luts, Jonathan Pool, Mike Rosner, Rolf Schwitter, & John Sowa. On controlled natural languages : Properties and prospects. In ***Controlled Natural Language***, pages 281–289. Springer, 2009. 58, 59

Matt Wynne & Aslak Hellesoy. ***The Cucumber Book : Behaviour-Driven Development for Testers and Developers***. Pragmatic Bookshelf, 2012. 67, 83

A Yurchyshyna, C Faron-Zucker, N Le Thanh, & A Zarli. Towards an ontology based approach for formalising expert knowledge in the conformity checking model in construction. In ***Proc. of the 7th European Conference on Product and Process Modelling (ECPPM)***, pages 10–12, 2008a. 49, 69

Anastasiya Yurchyshyna. ***Modélisation du contrôle de conformité en construction : une approche ontologique***. PhD thesis, Université de Nice-Sophia Antipolis, 2009. 9, 14, 17, 23, 28, 31, 45, 56, 86, 92, 165

Anastasiya Yurchyshyna, Catherine Faron-Zucker, Nhan Le Thanh, & Alain Zarli. Towards an ontology-enabled approach for modeling the process of conformity checking in construction. In ***CAiSE Forum***, pages 21–24, 2008b. 49, 69

Chi Zhang, Jakob Beetz, & M. Weisen. Interoperable validation for IFC building models using open standards. ***ITcon–Journal of Information Technology in Construction, Special Issue : ECPPM 2014***, 20 :24–39, 2015. 69