An Approach for Automatic Formalization of Business Rules

Cheikh Kacfah Emani, Catarina Ferreira da Silva, Bruno Fiès, Alain Zarli, Parisa Ghodous

An Approach for Automatic Formalization of Business Rules

Cheikh KACFAH EMANI, cheikh.kacfah@cstb.fr, chkacfah@liris.cnrs.fr
Centre Scientifique et Technique du Bâtiment (CSTB), Sophia Antipolis, France & Univ Lyon, Université Lyon 1, CNRS, LIRIS, UMR5205, F-69622, Lyon, France

Catarina Ferreira Da Silva, catarina.ferreira@univ-lyon1.fr
Univ Lyon, Université Lyon 1, CNRS, LIRIS, UMR5205, F-69622, Lyon, France

Bruno Fiès, bruno.fies@cstb.fr
Centre Scientifique et Technique du Bâtiment (CSTB), Sophia Antipolis, France

Alain Zarli, alain.zarli@cstb.fr
Centre Scientifique et Technique du Bâtiment (CSTB), Sophia Antipolis, France

Parisa Ghodous, parisa.ghodous@univ-lyon1.fr
Univ Lyon, Université Lyon 1, CNRS, LIRIS, UMR5205, F-69622, Lyon, France

Abstract
This paper presents an approach that aims to suggest to Construction experts a formal representation of given requirements. When available as formal expressions, requirements are suitable for automatic compliance checking. The goal of conformity checking is to answer the question “which components of a building project are non-compliant to a set of construction rules?”. When we consider both the size and the complexity of corpora of construction requirements, a computer-aided compliance checking process would be beneficial for experts. Such checking process requires a formal representation of building projects and construction rules. Nowadays, there are various tools for the formal designing of building products obeying to the Industry Foundation Classes (IFC) standard (e.g. Revit Building of Autodesk, ArchiCAD by Graphisoft, and ACTIVe3d by ARCHIMEN GROUP). On the other hand, to the best of our knowledge, there isn’t any tool which allows business experts to convert automatically and in a formal language, construction rules written in natural language. We propose an approach which intends to convert automatically natural language requirements into formal expressions. This approach relies on the IfcOWL ontology and represents formal rules as SPARQL queries using the RAINS language as an interim result. RAINS is a controlled natural language (CNL), whose sentences can be transformed automatically into SPARQL queries, and as a CNL it hides the complexity of formal languages.

Keywords: Business Rules, Conformity Checking, SPARQL, RAINS, Controlled Natural Language

1 Introduction

1.1 Motivations

Automatic conformity checking of projects is not new in the field of AEC. Many works in the literature have tackled one or more aspects of this task. We have for example works where recommendations are provided for the formalization and the organization of conformance requirements(Yurchyshyna and Zarli, 2009) (Bouzidi et al., 2012). We also have works in which authors propose approaches for modeling the conformity checking process(Yurchyshyna et al., 2007), and works where conformity checking is fully implemented(Yurchyshyna and Zarli, 2009),(Pauwels et al., 2011). In some other works, we

can find proposals for languages or formalisms to represent requirements; for example SPARQL (Yurchyshyna et al., 2007), N3 Logic (Pauwels et al., 2011), conceptual graphs (Solihin and Eastman, 2015), etc. In all these works, we have mainly two common features: (1) The transformation of requirements from their original forms (natural language texts, tables, figures) to the wished formal representation is *performed manually* (2) A business expert who aims to rewrite requirements into a formal expressions needs *the assistance of someone aware of formal representation languages or formalisms*. These two facts are the main causes of the lack of flaws detection functionalities in Computer Aided Design (CAD) tools and for those that offer conformity checking service, the number of the formal norms, to check against the mockup of building projects, is quite limited.

1.2 Contributions

Current approaches for the formalization of norms are manual and require the help of a knowledge representation expert. In this paper, we propose an approach for the automatic formalization of business rules. The approach takes as input a natural language (NL) sentence, expressing a single requirement, and a domain ontology and provides a RAINS rule(Kacfah Emani et al., 2016). RAINS is a controlled NL suitable to express conformity requirements. Any valid RAINS sentence is automatically rewritable as a SPARQL Rule. Hence the approach we propose aims to convert an NL requirement to a SPARQL Rule automatically. We apply this approach using the IfcOWL ontology (Pauwels and Terkaj, 2016), and we carry out a preliminary evaluation on norms extracted from the Norm (EN 81-41, 2011) on lifting platforms. To the best of our knowledge, it is the first automatic approach for the formalization of NL requirements in the field of AEC. As we will see, this approach, combined with the benefits of the language RAINS, provides a new paradigm for the building of formal rules repositories where the assistance of a knowledge representation expert is no longer an obligation.

2 Brief overview of conformity checking

Regulations are a set of rules which define the minimum level of acceptability and safety for construction projects (buildings and equipment). Conformity checking is a process (or a set of processes) which aims to detect components of a project which are non-compliant with regulations. With the growing adoption of Building Information Modeling (BIM), they are many stages that a construction project undergoes before the construction step itself (design, consultation, simulation, communication). At each of these stages, many investigations are performed to validate the digital mockup of the project before initiating the next stage. Among all these investigations, we find the validation of the project against the current regulations. This task is hardened by the complexity of the corpus of regulations. Indeed, regulations are described in documents of many types[1] (“arêtés”, “décrets”, European norms, local norms, etc.), with their custom strengths (some texts are obligatory and others stand as best practices, some texts have a higher priority than others, etc.) and topicalities (some texts are new and update or cancel existing ones, etc.). Also, the size of the regulations brings its difficulty to the problem. For example, French building codes cover more than 85.000 pages[2]. Consequently, an automatic checking of the compliance of building products with regulations is more than welcome. But to be able to rely on a computer to accomplish this task, we need to have a computer-readable version of regulations. This issue has led to many works which tackle one or more features of automatic conformity problems. We can classify these works into two main categories as illustrated by Figure 1A and 1B.

[1] All these characteristics reflect the organization of the French regulations.
[2] http://www.batipedia.com/

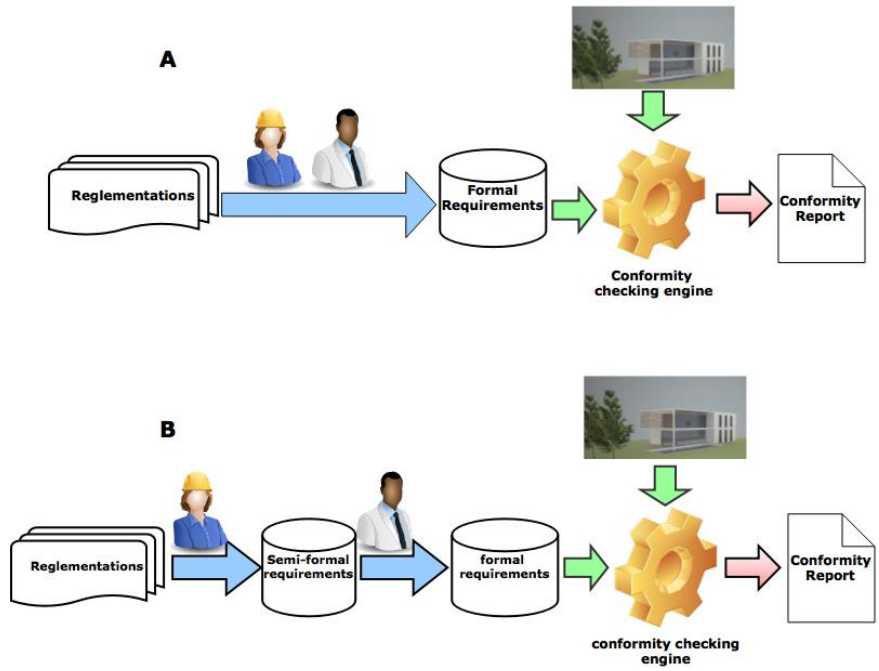

Figure 1 An overview of current approaches for the building of a repository of formal requirements for conformity checking purposes

The first picture shows the "traditional" and most adopted approach of the building of a repository of formal rules to be used in automatic conformity checking procedures. For example, works like (Yurchyshyna et al., 2007), (Pauwels et al., 2011) use this approach for feeding the storehouse of rules. We notice that this approach requires a synchronous collaboration between the Construction expert and the knowledge representation (KR) specialist. Moreover, in case of updates of some legal texts, even if the Construction expert can retrieve the formal requirements that require modification, he/she needs the support of someone with KR skills to perform the updates.

When we compare Figure 1A to Figure 1B, we see a change in the way formal rules are provided. Indeed, the business expert and the KR expert who work together in the first group of approaches can now work asynchronously. There is a break in the feeding of the formal requirements repository. Business experts transform original norms into a transitional disambiguated language like in (Bouzidi et al., 2012) with SBVR[3] or formalism like in (Solihin and Eastman, 2015) with conceptual graphs[4]. Then a KR expert relies on this interim representation to build the formal requirements. But once more, the support of another expert in addition to the business expert is compulsory. Also, even if the business expert can maintain the transitional repository by himself, all the changes he performs need to be transmitted by a person with KR skills.

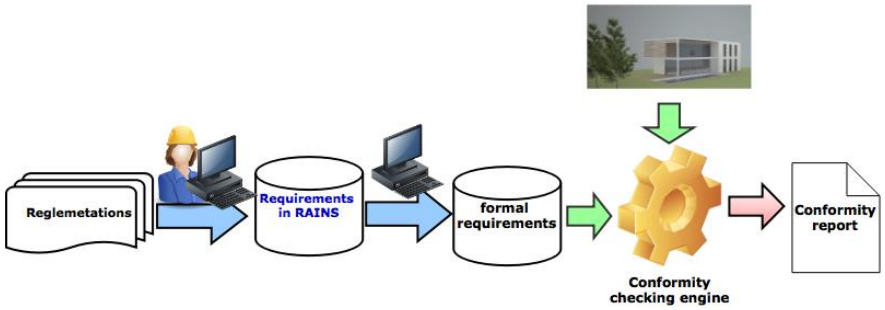

Figure 2 A new paradigm of the setting up of a repository of formal requirements for conformity checking purposes

[3] http://www.omg.org/spec/SBVR/
[4] http://conceptualgraphs.org/

To escape from this resources (time and people) consuming paradigm for the setting up of a storehouse of formal requirements, we propose a new paradigm, illustrated in Figure 2. We can see that there is still a transitional stage before having requirements in a formal representation. But there is only one type of actor. Construction experts only, take action in the rewriting of original rules. They need to write rules using an interim language or formalism that can be translated automatically into a formal representation. Even when updates occur, they can go back into the transitional repository and modify requirements and the resulting update rules will be transformed automatically into formal expressions. Of course, if this language is new to the experts, they must take a certain time to learn this representation that has the obligation to hide the complexity of KR formalisms or languages.

In this work, this transitional language to be used by Construction experts is the controlled natural language called RAINS (Kacfah Emani et al., 2016). We give an overview of this language in the next section before presenting our approach of formalization of text rules and that directly relies on RAINS.

3 RAINS: a controlled natural language for conformity rules

3.1 Goals and motivations

Automatic compliance checking of building products is a widely discussed topic in the AEC community. This task mainly relies on the availability of requirements in a computer-readable form. But the transformation of the actual norms into formal requirements is time-consuming and requires a double competency in the field of Construction and in KR. We argue that we can do without KR skills or experts if construction engineers can directly rewrite the rules in a semi-formal manner closed to the natural language they master. This assessment leads to the proposal of a controlled natural language (CNL), called RAINS, for the specification of requirements (Kacfah Emani et al., 2016).

A CNL, a.k.a. Controlled Language (CL), is a language that is *explicitly built*, and that is supported by an existing language. A CNL has a *grammar*, *vocabulary,* and *expressiveness* that are restricted in comparison to the natural language on which they take advantage (Kuhn, 2014). In some cases, a CNL has a style that allows it to be completely interpretable by a computer.

It is the case of RAINS which can be parsed and transformed automatically into a SPARQL query (see Section 3.5) and that is the main goal of this language: to get rid of a KR expert after the work of rewriting and disambiguation performed by Construction experts. Every CNL needs a vocabulary. In the case of RAINS, this vocabulary is made of (1) very few language keywords (*if*, *then*, *and*, *not*, etc.) which allows the experts to write requirements in rule-like statements (2) some comparators (equal, greater than, etc.) and (3) mainly labels of the Industry Foundation Classes (IFC) entities (see Section 3.3) . When we will look at some examples of RAINS sentences, we will notice that it is a language which mainly requires from the Construction experts that they understand clearly what requirements state and they must be aware of the IFC schema[5].

In this paper, the referenced IfcOWL ontology is the one provided by (Pauwels and Terkaj, 2016) and enriched with some entities found in the lifting platform Norm (EN 81-41, 2011), and annotation entities[6] (Yurchyshyna and Zarli, 2009). Figure 3 depicts an excerpt of this ontology. It is important to mention that, for the next lines, we assume that the reader is familiar with Web Ontology Language[7] (OWL), Resource Description Framework (RDF) and RDF Schema (RDFS) terms and axioms (*entity, concept, class, property, object property, datatype property, individual, range, domain*, etc.)

[5] http://www.buildingsmart-tech.org/ifc/IFC4/final/html/

[6] Annotations are introduced in the work of (Yurchyshyna and Zarli, 2009) to describe formal requirements for automatic conformity checking goals. For instance, we can annotate a rule with the value “individual house” for the “building use” feature. Hence this rule will not suit the compliance checking of a “public building” or a “collective housing”.

[7] https://www.w3.org/TR/owl-guide/

3.2 Assertions in RAINS

Before presenting the formal syntax of RAINS, we find important to present what we call *assertion* in RAINS. RAINS is designed in a way which forces the writer to break down a requirement in a simple set of *atomic pieces*. For example, saying that "door's width must be between 80 and 100 cm" means that: (1) "a door's width must be greater than 80 cm" and that (2) "the width of this same door must be less than 100 cm." We call each of these atomic pieces of information when put in RAINS syntax, *an assertion*. There are five (05) types of assertions in RAINS specification. We present them at a glance and provide an example of each of them. These examples are valid RAINS rule (w.r.t. to the formal syntax of RAINS – Section 3.3). To foresee the syntax of RAINS, let us mention that a RAINS rule necessarily follows the logical pattern (*If … then*) and assertions use *RAINS entities* (comparators, negation markers, entities found in IFC, etc.) easily identifiable by *quotation marks* (" ").

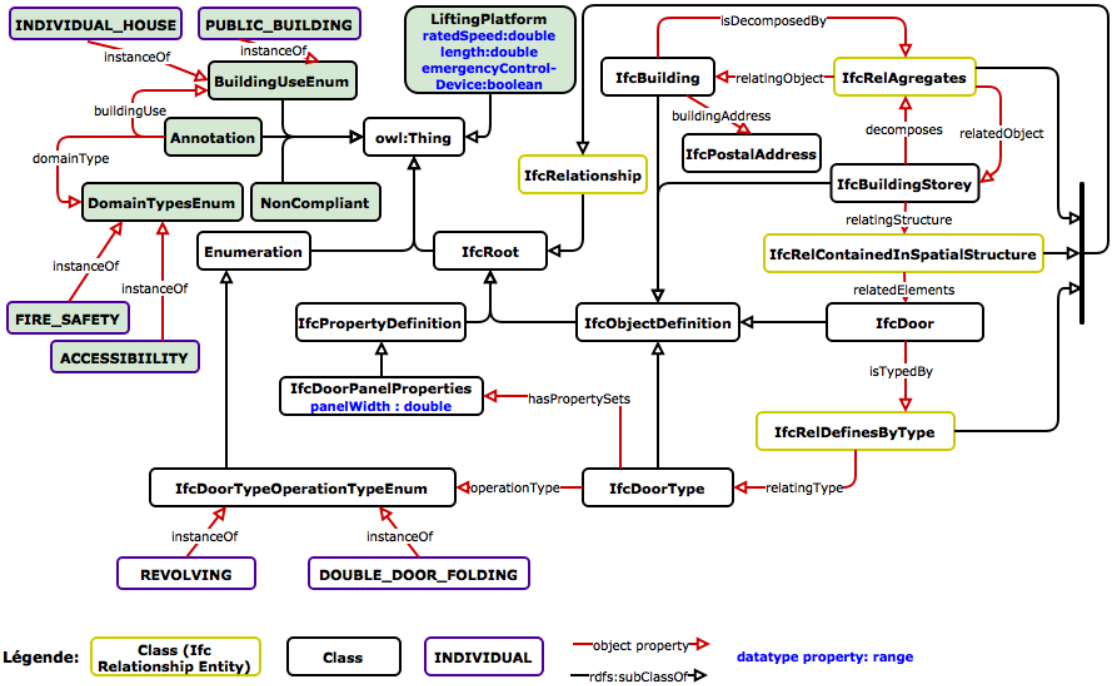

Figure 3 Part of the IfcOWL Ontology **(Pauwels and Terkaj, 2016)** enriched with new entities (with a green background)

Here are the five types of RAINS assertions, followed by two examples of RAINS rule to illustrate them.

- *Single concept assertion*. It is an assertion built on a single concept (from the domain ontology) – see example 1 **(SCA)**.
- *Double concept assertion*. In this case, assertion is built on two concepts of the domain ontology and no additional entities from the ontology – see example 2 **(DCA)**.
- *Data assertion*. It is an assertion which allows the description of a concept using a *datatype property* – see examples 1 and 2 **(DA)**.
- *Boolean assertion*. It is a subtype of data assertion where the datatype property has a Boolean range – see example 1 **(BA)**.
- *Object assertion*. In this kind of assertion, we find two concepts (or a concept and an individual) related by an object property – see example 2 **(OA)**.

Example 1

If the "rated speed" of a "Lifting platform" is "greater than" "0.15" m/s **(DA)**
And an "emergency control device" is "not" found in this "Lifting platform"2 **(BA)**
Then it is "Non-Compliant"2 **(SCA)**

Example 2

Given an "Ifc door type" that "has property sets" an "Ifc door panel properties" **(OA)**
If this "Ifc door panel properties"3 has a "panel width" that is "less than" "80" cm **(DA)**
Then this "ifc door type" is "Non-compliant"1 **(DCA)**

3.3 Formal syntax of RAINS

Having a formal syntax is a prerequisite for a CNL to envision the automatic formalization of its sentences. We use the Extended Backus-Naur form to describe the syntax of RAINS[8]. See Table 1 for the meaning of the symbols of this notation.

In Listing 1 we see that an extract of the formal syntax of RAINS. We can notice that a RAINS rule is made of an *antecedent* and a *consequent* and *available annotations* (line **1**).

Table 1 Some Extended BNF symbols and their meaning

Symbols	Meanings
Unquoted words	Non-terminal symbols
'…'	Terminal symbol
\d	A digit
+	One or more times the preceding expression
*****	Zero or more times the preceding expression
?	Zero or one occurrence of the preceding expression
…Description…	A textual description of symbol
=	Defining
,	Concatenation
|	Alternatives

Listing 1 Extract of the formal syntax of RAINS **(Kacfah Emani, et al., 2016)**

```
1  Rule = Annotation*, Antecedent, Consequent;
2  Annotation = '@', AnnotationName, '(', AnnotationValue+, ')';
3  Antecedent = ((' If'|' Given'|' When'|' But'|' And'), Assertion)+;
4  Consequent = ' Then', Assertion, (' And', Assertion)*;
5
6  Assertion = ( SingleConceptAssertion | DoubleConceptAssertion  | DataAssertion | BooleanAssertion | ObjectAssertion ), CarRetrn;
8  SingleConceptAssertion = Text?,  Concept, Text?;
9  DoubleConceptAssertion = Text?,  Concept, Text?, Neg?, Text?,  Concept;
10 BooleanAssertion = OrderedBooleanAssertion | InvertedBooleanAssertion;
11 OrderedBooleanAssertion = Text?,  Concept, Text?, Neg?,  BooleanDatatypeProperty, Text? ;
12 InvertedBooleanAssertion = Text?,  BooleanDatatypeProperty,  Text?, Neg?, Text?,  Concept, Text?;
14 DataAssertion = OrderedDataAssertion | InvertedDataAssertion;
15 OrderedDataAssertion = Text?,  Concept, Neg?, ChainOfObjectProperties?,  DatatypeProperty, Comparator, Quantity, Text?;
17 InvertedDataAssertion = Text?,  DatatypeProperty, Text?,  Concept, Text?,  Neg?, Comparator, Quantity, Text?;
19 ChainOfObjectProperties =  ObjectProperty ,(' .',  ObjectProperty)*;
20 Concept = ' " ', ConceptLabel, '"', Integer?;
21 Text = ...free text in natural language... ;
22 CarRetrn = ...the carriage return char... ;
23 Quantity = Literal | Text?,  DatatypeProperty, Text?,  Concept;
24 Literal = Integer | Float | String | Boolean;
25 Neg = ' "no"'|' "not"'|' "is not"'|' "do not"'|' "does not"';
26 Comparator =  "equals"'|"is equal to"'|' "less than" '| ' "is less than" '|' "greater than" ' | ' "exceed" ';
```

Also, antecedent and consequent are made of a list of assertions, each introduced by appropriate RAINS keyword (lines **2**-**3**). These assertions are of a certain type (line **6** and Section 3.3). For each type of assertion, we have its definition[9], and we see each assertion relies on domain ontology entities (concepts, datatype and object properties, individuals)

[8] https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_Form

[9] For a complete list of the definition of assertions, please see (Kacfah Emani et al., 2016).

through their labels (see for example line **20**). The optional number after a concept label denotes co-reference as we illustrate in the next section. Negations markers and built-in predicates like comparators can be used in assertions too. We give more explanations of this syntax using examples in the next sections.

3.4 A detailed example of a RAINS rule

The rule "*In buildings with public access, the platform length shall not be less than 1 400 mm, to enable sufficient space for an attendant*" (EN 81-41, 2011) can be written in RAINS like in A or B (or other).

A @ Building use (public building)
If the "length" of a "Lifting platform" is "less than" "1400" mm
Then it is "Non-Compliant"2

B @ Building use (public building)
Given a "Lifting platform" having a "length" that is "less than" "1400" mm
Then this "Lifting platform"1 is "Non-Compliant"1

In the RAINS rule A, we have the antecedent, introduced by the keyword **If,** that has a single assertion that is a data assertion. This data assertion is made of four (4) RAINS entities, that are identifiable by the quotation marks " ". We notice that in a RAINS rule, *concepts start with a capital* (for example "Lifting platform") and all the *other type of entities start with lower case*. As the formal syntax of RAINS says, in a RAINS rule, we are free to use free NL expressions (line **21** of Listing 1) to link entities and to make a RAINS sentence looks natural. Also, these natural expressions can be used to clarify information found in the rule; for instance, the NL phrase mm after "1400" tells the reader the unit of this value, even if this unit is not part of RAINS entities. Next, we have the consequent of the rule, introduced by the keyword **Then**, and which consists of a single concept assertion. In this assertion, we found the concept "Non-Compliant"2, *where the number 2 refers to the 2nd entity of the rule*, "Lifting platform." Finally, we notice an annotation which tells that *this rule is only suitable if we are checking the compliance of a platform located in a building of public access*.

In the RAINS rule B, we have another RAINS version of the same requirement, and we see that we can introduce the antecedent using keyword different from 'If', or use an *ordered data assertion* (Listing 1 - line 15) instead of an *inverted data assertion* (Listing 1 - line 17). This choice leads to a different number to resolve the reference of entities of the consequent ("Lifting platform"1 and "Non-Compliant"1) to the preceding mention of "Lifting platform." But these two RAINS rules *have the same meaning*.

3.5 Formal Semantics of RAINS

We kindly refer the reader to the dedicated section of (Kacfah Emani et al., 2016) for the full semantics of RAINS. Every RAINS rule can be directly converted into an annotated SPARQL query. We already mention that annotations are nothing more than descriptive information that can be used in the conformity process itself. The SPARQL query derived from a RAINS rule relies on assertions found in the antecedent and the consequent and only on entities (i.e. phrases between " ") of the rule i.e. NL phrases in a RAINS rule are meaningless for the parser. If we suppose that the *prefixes* (rdf:, etc.) shared in the Semantic Web (SW) community are defined[10], the above RAINS rules provide:

```
1 :AnnotationRule1 rdf:type :Annotation ; :buildingUse :PUBLIC_BUILDING.
2 :Rule1 rdf:type :Rule; :hasAnnotation :AnnotationRule1;
3       :hasQuery  "CONSTRUCT{?platform rdf:type :NonCompliant}
4                   WHERE{ ?platform rdf:type :LiftingPlatform.
5          FILTER NOT EXISTS {?platform :length ?lg. FILTER(!(?lg < 1400))}} "
```

At line 1 we have an instance of :Annotation with the given value. At line 2, we have a :Rule and we link it to the adequate instance of :Annotation. Lines 3-5 provide the body of the SPARQL conformity rule. The antecedent leads to the CONSTRUCT clause and the consequent to the

[10] See http://prefix.cc/ for the extended version of prefixes shared by SW people

WHERE clause. The SPARQL rule says "we conclude that a Lifting platform is non-compliant if we cannot assert that it has a length that is not less than (i.e. greater than or equal to) 1400". In this case, *even if the length of a platform is not available*, this platform will be considered faulty.

4 Automatic formalization of business Rules

The goal of the approach we present here is to obtain a RAINS rule from an NL requirement. Since a RAINS rule can be converted automatically into a formal rule, we consider that this approach automatically formalizes requirements. We choose to obtain a RAINS sentence i.e. a sentence in CNL, because in practice, it can be shown to a Construction expert for edition/validation. Hence, he/she can correct the errors that occur during formalization. In the next sections, we present the steps of our approach and illustrate each step with the following requirement "*In buildings with public access, the platform length shall not be less than 1 400 mm, to enable sufficient space for an attendant*".

4.1 Assertions identification

A RAINS rule is made of atomic and meaningful assertion. Our first step consists in finding the potential assertions in our NL requirement. This task is known in natural language processing (NLP) as *Open Information Extraction*. Here we use the CSD-IE tool[11] (Bast and Haussmann, 2013) and we obtain three NL assertions from our example:

- A1: "*the platform length shall not be less than 1 400 mm"*
- A2: "*In buildings with public access*"
- A3: "*to enable sufficient space for an attendant*"

4.2 Natural phrases detection and alignment

From each of the NL assertions, we use a *phrase chunker* to identify groups of words that go together. In this approach, we consider, *heuristically*, that each group of word hints at least a formal entity. We use the Illinois chunker (Punyakanok and Roth, 2001) for this task and obtain:

- A1: "NP[*the platform length*] VP[*shall* not_N *be*] NP[*less* $than_K$ 1400_L *mm*]*"*
- A2: "PP[*In*] NP[*buildings*] PP[*with*] NP[*public access*]"
- A3: "VP[*to enable*] NP[*sufficient space*] NP[*for an attendant*]"

NP, VP, and PP stands for noun, verb and prepositional phrases. To look for entities, *we discard PPs*. Then, in each NP and VP: (1) We identify negation markers (not_N), comparators ($less\ than_K$), literals (1400_L) and then ignore them for alignment of phrases with the ontology; (2) From remaining tokens, we prune stop words (modals, determiners, etc.) which are in *black color* in the example above. Next,

- From non-empty VPs, we find the *property* (datatype or object) of the ontology, whose label has the highest similarity score with the remaining tokens (in *blue*) of the VP. This score must be higher than a certain threshold unless we simply consider that this VP hints an object (**O**) or datatype (**D**) property, that is still unbound.
- From non-empty NPs, we find the property *and* the concept/individual of the ontology whose label matches the most the remaining tokens (in *blue*) of the NP. We tag the property with the letter **O** or **D** depending whether it is an object or datatype property. Same thing with the letters **C** and **I** if we get a concept or an individual. Here we also have a threshold which plays the same role as in matching of VPs; but here *we look in entities both in each noun token and in all the tokens in an NP*.

When we apply these instructions on the phrases of our example, we get:

- A1: {:LiftingPlatform$_C$, :length$_D$, not_N, $less\ than_K$, 1400_L, unbound:mm$_X$} or {unbound:PlatformLength$_X$, not_N, $less\ than_K$, 1400_L, :mm$_X$} …
- A2: {:IfcBuilding$_C$, unbound:PublicAccess$_X$}

[11] http://metropolis.informatik.uni-freiburg.de:6543/

- A3: {unbound:enable$_X$, unbound:SufficientSpace$_X$, : unbound:Attendant$_X$}

Let us mention that we use the N-gram string similarity measure[12] (with n = 2) for string matching and that we heuristically take the threshold at 0.5.

4.3 RAINS pattern resolution

We use the tag assign to each entity to get a string that we call a pattern. For example the ordered list of entities {:LiftingPlatform$_C$, :length$_D$, *not*$_N$, *less than*$_K$, *1400*$_L$, unbound:mm$_X$} gives the pattern CDNKLX[13]. RAINS assertions also have their patterns, provided by their formal syntax. For instance, if we consider the definition of a data assertion (Listing 1 – line 14), we see that it can be either an ordered or inverted data assertion. And when we look at the sequential list of tags of entities required for each of these subtypes of assertions (lines 15-16 and line 23), we get: (1) for ordered data assertion {CDKL, CODKL, CDNKL, CODNKL, CDKDC, CODKDC, CDKNDC, CODKNDC} (2) for inverted data assertion {DCKL, DCNKL, DCKDC, DCNDC}. So, the union of these two lists of patterns composes the patterns of a RAINS data assertion.

At this stage, *for each NL assertion*, we need to answer the question "which pattern of RAINS assertions fits the best to the patterns of this NL assertion?". This answer is given using the voting algorithm given in Listing 2.

Listing 2 Resolution of the best RAINS pattern for each NL assertion (keywords, *functions*, //comments). For the levenshtein distance see https://en.wikipedia.org/wiki/Levenshtein_distance.

```
1 For each assertion in nlAssertions
2    setOfbestRAINSPatterns = { };      bestRAINSPatterns = { };
3    For each assertionPattern in listOfPatterns(assertion) // we discard patterns with more than one X
4    maxScore = - maxInt; bestRAINSPatterns = { };
5    For each rainsPattern in rainsPatterns
6       score = - levenshteinDistance (assertionPattern, rainsPattern);
7       If (score > maxScore) then maxScore = score;
8    End for
9    For each rainsPattern in RAINSPatterns
10      If (-levenshteinDistance(assertionPattern,rainsPattern) = maxScore) then
11         bestRAINSPatterns = bestRAINSPatterns U rainsPattern;
12   End for
13      setOfbestRAINSPatterns = setOfbestRAINSPatterns U bestRAINSPatterns;
14   End for
15   bestRAINSPattern = rainsPatternWithMostOccurrences(setOfbestRAINSPatterns); //when many patterns
have the highest occurrence, we heuristically choose one between those having a property
16End for
```

For our running example, we have (remember that we discard pattern we more than one X):

- A1: list of patterns {CDNKLX, ~~XNKLX~~, …}; set of best RAINS patterns {{CDNKL}, { }}; best pattern CDNKL
- A2: list of patterns {CX}; set of best RAINS patterns {{CC, C, CB}} best pattern CB (the two others patterns, C and CC do not have a property – Listing 2, line 15)
- A3: list of patterns {~~XXX~~} set of best RAINS patterns {{ }}; best pattern (None)

4.4 Main predicate and main concept identification

Having the main predicate of a requirement helps to write a requirement as a non-conformity query and the main concept serves for the construction of the consequent of the rule. We identify both of them using the dependencies between the tokens of the original NL requirement. We used the Stanford parser[14] for this purpose. With our running rule, we obtain the graph of dependencies shown in Figure 4. We see that the *root* of our sentence is the token "*mm*" (millimeter).

[12] http://dl.acm.org/citation.cfm?id=2179011
[13] Tags meaning: C = Concept, D = Datatype property, O = Object property, L = Literal, N = Negation mark, K = Comparator, I = Individual, X = unbound entity, B = Boolean datatype property
[14] http://nlp.stanford.edu:8080/parser/index.jsp

- The main predicate is an *NL word* that is the root if this root is a verb, otherwise, is the *copular* (dependency name *cop*) to the root. In this case, our main predicate is the verb "*be*".
- The main concept is the *formal concept entity* that we obtained using the nouns related to the root via a subject (*nsubj* or *nsubjpass*) dependency. In our example, from the root "*mm*", we can reach, in order, the two nouns (remember that only nouns can lead to concepts – Section 4.2) "*length*" and "*platform*". "*length*" was aligned with the datatype property :length and "*platform*" with the concept :LiftingPlatform that is hence the main concept of our rule.

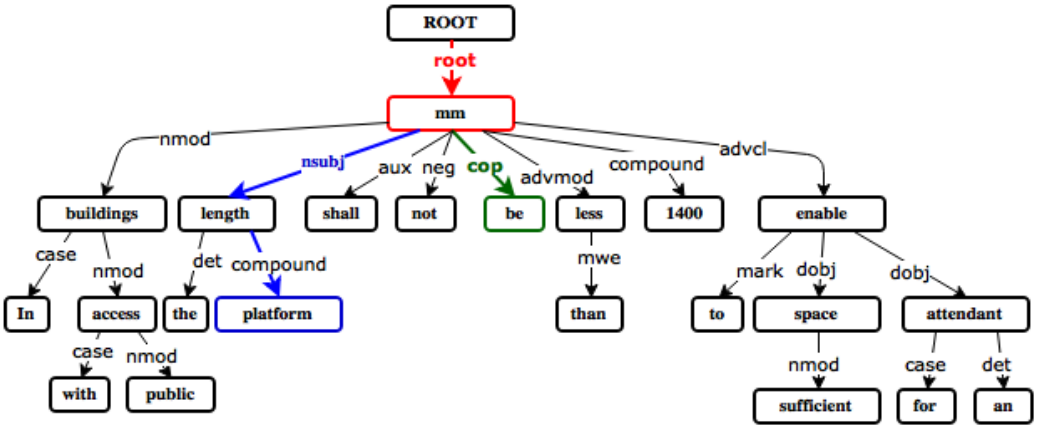

Figure 4 Graph of dependencies of the running requirement

4.5 Recombination (antecedent resolution)

The goal of this step is to put together the set of RAINS assertions we obtained at the preceding stage and to link them taking advantage of common entities. These assertions compose the antecedent of the rules as we can see it on examples of RAINS rules (see Sections 3.2 and 3.4). We illustrate this step using our running example. For this purpose, we need our assertions and their entities (Section 4.2), the RAINS pattern of each assertion (Section 4.3) and the main predicate and the main concept of the rule (Section 4.4).

- A1: **entities** {:LiftingPlatform$_C$, :length$_D$, *not*$_N$, *less than*$_K$, *1400*$_L$, unbound:mm$_X$}. **RAINS pattern**: CDNKL. The pattern requires nothing more than (in this order) a concept, a datatype property, a negation marker, a comparator and a literal. Hence, the last unbound entity is noise for getting a valid RAINS assertion. **Resulting RAINS Assertion:** **If** "Lifting platform" "length" "not" "less than" "1400"
- A2: **entities** {:IfcBuilding$_C$, unbound:PublicAccess$_X$}; **RAINS pattern** : CB. This pattern (with the tag B) hints that the unbound entity should be a boolean datatype property. **Resulting RAINS Assertion:** **And** "IfcBuilding" "publicAcess"
- A3: useless since no RAINS pattern suits this assertion (see Section 4.3)

At this step, we have two RAINS atomic assertions and we notice that they are *independent* i.e. not common concept between the two. Heuristically, we keep the one having the main concept, since this main concept will be part of the consequent. So, only the first assertion is still useful for our approach.

Finally, we use the main predicate to make explicit the non-conformity feature of our rule: for all the RAINS assertions that were derived from an NL assertion having the main predicate (we recall that the main predicate is an NL word) (1) if they have a negation marker, we delete this marker (2) if not, we introduce a negation marker (in accordance with patterns of RAINS assertions[15]). To illustrate this idea let's take the sentence "*You must have an X*" the main predicate is "*have*". To highlight non-compliancy, we can write it like "if you don't have an *X then you are in trouble*". Likewise, "*You shouldn't have a Y*" could be rewritten "*if you have a Y then you are in trouble*".

[15] We can notice in Listing 1that all the assertions forecast the use of a negation (Neg?). So we know, depending on the pattern of a given assertion where to add the negation mark.

Applied to our single remaining assertion, we get: **If** “Lifting platform” “length” “less than” “1400”.

4.6 Consequent resolution

The consequent of our RAINS rule can always be **Then** is this “Non-Compliant”n, where n is the rank of the main concept in the ordered list of entities found in the antecedent. Here *n equals 1*.

Finally, the RAINS rule we get is:

If “Lifting platform” “length” “less than” “1400”
Then is this “Non-Compliant”1

5 Concluding remarks and future work

In this paper, we present a heuristic-based approach for the automatic formalization of rules. It is the first automatic approach, to the best of our knowledge, in the field of AEC for the formalization of rules. This approach takes an NL rule and provides a RAINS version of this rule. RAINS is a CNL suitable to write conformity requirements. Every RAINS sentence is made of atomic assertions and can be converted automatically into an annotated SPARQL query. Actually, the approach we propose is able (1) to identify all the atomic assertions for the formalization of a rule (2) to formalize assertions having no property or having datatype or object properties (3) to prune meaningless (w.r.t. formalization) pieces of information (4) to rewrite rules in a form suitable to detect flaws (5) and to provide a result understandable by a Construction expert (which is aware of RAINS) and we can correct it in case of errors. We see that the approach does not propose a way to find potential annotations within the NL requirement. When we see the result of our running example, we see that we miss identifying the annotation @ **Building use (public building)**. Also, although the resulting RAINS rule is valid w.r.t the formal syntax of RAINS, it lacks some NL phrases to improve the naturality of the result. Finally, we will set up a corpus and define metrics and evaluate our approach against that test material.

References

Bast, H., Haussmann, E., 2013. Open information extraction via contextual sentence decomposition, in: Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on. IEEE, pp. 154–159.

Bouzidi, K.R., Fies, B., Faron-Zucker, C., Zarli, A., Thanh, N.L., 2012. Semantic web approach to ease regulation compliance checking in construction industry. Future Internet 4, 830–851.

EN 81-41, 2011. Safety rules for the construction and installation of lifts - Special lifts for the transport of persons and goods - Part 41 : vertical lifting platforms intended for use by persons with impaired mobility. Assoc. Fr. Norm. AFNOR.

Kacfah Emani, C., Ferreira Da Silva, C., Fiès, B., Bourdeau, M., Ghodous, P., 2016. RAINS: A controlled natural language for conformity rules (Technical).

Kuhn, T., 2014. A survey and classification of controlled natural languages. Comput. Linguist. 40, 121–170.

Pauwels, P., Terkaj, W., 2016. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. Autom. Constr. 63, 100–133. doi:10.1016/j.autcon.2015.12.003

Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, J., De Meyer, R., Van de Walle, R., Van Campenhout, J., 2011. A semantic rule checking environment for building performance checking. Autom. Constr. 20, 506–518.

Punyakanok, V., Roth, D., 2001. The use of classifiers in sequential inference, in: NIPS. MIT Press, pp. 995–1001.

Solihin, W., Eastman, C., 2015. Classification of rules for automated BIM rule checking development. Autom. Constr. 53, 69–82.

Yurchyshyna, A., Faron-Zucker, C., Le Thanh, N., Lima, C., Zarli, A., 2007. Towards an ontology-based approach for conformance checking modeling in construction, in: PROC. OF «24TH W78 CONFERENCE–BRINGING ITC KNOWLEDGE TO WORK. pp. 195–202.

Yurchyshyna, A., Zarli, A., 2009. An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction. Autom. Constr. 18, 1084–1098.