Response to Reviewers

Review #1

1. One of the announced contributions of this article is the Structured Natural Language (SNL) that has been described in section 3. However, converting controlled nature language to computable rules is not a novel idea, hence it is important to see how expressive and how intuitive the language is. The introduction in section 3 needs to be elaborated and many questions need to be answered. How expressive this language is? Which forms and expressions in SPARQL it supports? How much percentage of rules can be represented by it? The current presented language seems simple and many useful constructs in rules e.g. quantifiers, cardinality, aggregation operators do not exist. There are many existing research and development to convert (controlled) natural languages to computable rules and queries in the building industry as well as in the Semantic Web field. They are all ignored by this article. For example, RASE from Hjelseth and Jiansong Zhang's research in the BIM field. There are also existing controlled natural languages like SBVR and SQUALL that either have been standardized or already have conversion patterns to SPARQL. Why not reuse them? Authors should prove the novelty of their approach.

Response: (interpretation of the expressiveness of the SNL language)

Currently, SNL is able to cover more than 85% of the national building codes (e.g. National Architectural Design Code for Fire Protection: 101/112) and more than 95% of the domain codes. There are two basic types of SNL sentences, simple sentences are used to describe the basic rules, such as all bedrooms have windows, or there is a bedroom its area < 20 and so on; conditional sentences are used to describe complex rules with conditional judgments. Since the building codes is usually a constraint on the properties and relationships of the building components, the SNL is mainly designed for both. The properties of the components can be described by "component its property". The relationships of the components is determined by "has", "distance between" and so on. The data types supported by SNL are mainly composed of strings and numbers. Since the Boolean type exists as a grammatical element in the natural language, the Boolean type is implemented by the keyword "no" or "not". Other data types (such as date, time, etc.) in the building code has almost no application, so SNL temporarily did not achieve them.

We have modified and supplemented these interpretations at the 5th paragraph in Section 3.

Response: (interpretation of SNL and the NLP based methods)

The SNL language is a structured natural language. Due to the inaccuracy of the semantics of natural language, many of the Building codes which are written in natural language need further interpretation by domain experts. As a result, it is difficult to translate the codes directly into correct rules by NLP techniques. We still need a good formalization to represent rules of building codes. Therefore, we propose a semi-automatic method. First

manually understand the building code, then use SNL language to enter the rules, and build a rule library for further checking.
SNL is a rule description language for building codes. As an intermediate language, it can transform natural language building codes into a SPARQL statement and applied directly to the model check. Compared with the RASE method, the SNL does not directly deal with the natural language specification, but the rules of the SNL statement are written by understanding the building code. Based on the similarity between the SNL and the natural language, the SNL statement is readable for the user and easy to understand, the rules will not contain labels and other information independent of the specification itself; and SNL can automatically generate SPARQL statement, no need to do re-logical reorganization or other processing.
This part has been added as the 2rd paragraph in Section 3.

Response: (Comparison of SNL and other controlled Natural language)
From this point of view, SNL function and SQUALL is somewhat similar, the difference is, SQUALL only package for SPARQL, writing SQUALL statement will still use the concept of RDF, which means exposing the intermediate data in the model checking process to the user who writes the rules, it is not only difficult for the user to understand but the exposures of such data are not needed. The level of SNL is higher, its writing logic and conceptual description are more similar to the natural language, the user only need to understand the concept of natural language on it, do not need to understand the knowledge of the semantic network, and SNL statement can also be correctly mapped to SPARQL, more suitable than the SQUALL for building model check. In other words, even if we use the SQUALL, we still need to provide users with SNL language to input the rules.
SBVR is mainly used to describe business rules, but there are many special cases in the building codes which are not easy for business rules description language to deal with, such as the reference between different building codes, or the terms may have a variety of conditions, different circumstances has different requirements, etc. At the beginning of the design, SNL is for the building code, taking the special aspects of the field of construction expertise and domain knowledge into account. Therefore, SNL is more suitable than the SBVR for building model checking.
This part has been added as the 3rd paragraph in Section 3.

2. In section 4, an approach to extract IFC subsets according to rule library is introduced. However, some steps of the extracting procedure are not clear and need additional explanations.
2.1. It seems related entities and attributes are derived from the rule library. How are all the terminologies and concepts used in the rule library structured and how are they mapped to IFC constructs?
Response: The components and attributes in the rules are similar to those in the natural language. By analyzing the SNL statements in the rule library, we can get the components and attributes of the SNL description and apply them to the IFC extraction process.
The concepts in the rule are mapped to the components in the IFC through a configuration file (.cfg). As to the different mapping from the concepts in the design codes to a concrete

BIM model representation, only the configuration is needed to modify. For example, the mapping defines the concept “Bedroom” in the rule library is defined in the IFC files with the entity “IFCSPACE” and its LongName string contains the substring “Bedroom”.
This interpretation has been added as the 2rd paragraph in Section 4.

2.2. In the second paragraph, “For the rules related to geometric computation, we need to extract the information such as…” how do you judge whether a rule is related to geometric computation?
Response: By sorting out the existing specifications, we have extracted the commonly used computational keywords, such as "distance", "vertical net distance" and so on. The rules with these keywords must be geometrically calculated to check. By matching these keywords in the process of SNL semantic parsing, we can see which rules need to be geometrically calculated.
This interpretation has been added at the 3rd paragraph in Section 4.

2.3. “For any r belonging to the RI, we extract the attribute set that belongs to the set A and referencing r (for example IFCRELDEFINESBYPROPERTIES) to get AI”. This needs additional explanations. There are many different IFC constructs associated with IfcRelationships. Propertysets, types, materials, classifications all have different structures, and they cannot all be simply derived by just extracting attributes from IfcRelationships.
Response: This is the case. We just take “**IFCRELDEFINESBYPROPERTIES**” as example. The relationships with types, materials are also considered, if the design codes have the constraints on them.
This interpretation has been added at the 5th paragraph in Section 4.

2.4. Are the extracted IFC sub models still valid IFC models?
Response: The answer is no. On the one hand, we will directly extract the IFC elements into OWL elements, theoretically we do not generate an extracted IFC model. On the other hand, since the IFC submodule is not our goal, our goal is to extract the information from it for subsequent checks, and it is not necessary to combine all the extracted IFC elements for further processing to be a legal IFC model.

3. In section 4, the extracted IFC sub models are then transformed to simplified OWL models. Again some existing research that systematically simplifies ifcOWL graphs is ignored by this research such as IfcWoD ontology and SimpleBIM ontology. There are also questions need to be answered. How many relationships have been shorten? How different the OWL graphs are before and after the simplification process? How modular and flexible this simplification process is, can the same program be reused for other rule cases?
Response: Thank you. The existent research like IfcWoD and SimpleBIM are really related to the extraction part of our work. They provide several approaches which make great simplification of the RDF graphs. We appreciated the approaches very much. The ideas like building direct relations between different concepts are very similar with our methods.

In this part, our method is not novel. However, there is one main difference between our work and the former simplification work. The goal of our work is regulation checking, thus the extraction procedure is regulation oriented. For example, if the regulation contains the geometry related items to be checked, the geometry information should be extracted, and cannot be simplified by the proposed methods in SimpleBIM. On the other hand, the properties which are irrelevant to the regulation can be deleted safely.

The related work are referred and the interpretation is added in the 2rd paragraph in Section 4.

The type of shortened relationship is relatively small, but because of the quantity of such kind of relationship in a model is very big, so the number is still a lot. In the actual process, the direct generated OWL map is a simplified graph, not firstly generate a non-simplified map, and then do the simplify work, so in fact we are not doing OWL map optimization work. The same program can be definitely reused by other rules. It is an automatic simplification process.

4. In section 5, three examples are presented to show how rules represented in the SNL language are transformed to SPARQL queries. It seems the transformation algorithm is very case specific. IFC can represent information in various ways, for example, the concept "Bedroom" can be represented by many different attributes and relationships. Authors should explain how to consider such problem. There are also some minor issues or mistakes in this part.
In the first example, the "Has" relationship between a space and a window is automatically recognized and converted to "hasBoundaryElement" as explained in this section, but in the third example, the "Has" relationship between a space and a window is converted to "isContaining". Some of the relation terms are confusing e.g. the "hasSubType" relationship between a building and a building storey.
Response: **(Explain to the concept mapping from SNL to SPARQL)**
As the answer to the question 2.1, the concept mapping from a BIM model to the extracted OWL model is based on the configuration file. The concept mapping from SNL to SPARQL is based on the same configuration file. Therefore, we can implement the SNL to SPARQL conversion process automatically. SNL statements that conform to the SNL syntax can be converted to SPARQL statements. For example, the concept "Bedroom" in the SNL sentence corresponds to the $IFCSPACE$ entity with its LongName containing the word "Bedroom".
This interpretation has been added at the 3rd paragraph in Section 5.

Response: **(Explain to the relation conversions from SNL to SPARQL)**
This is a feature of SNL to be a language to describe design codes. The "has" relationship at the design codes description level, can be converted automatically into different concrete model implementations like the ones in IFC files. We apply the domain knowledge of building to fulfill the automatic transformation. The different types of conversions for "has" are determined by the subject and object type of it. The "has" relationship between the

building and the storey is converted to "hasSubType". The other type of "has" relationship can be "isContaining", "hasBoundaryElement", or “isConnectedTo”, etc. For example, The “Has” relationship with the subject “BedRoom” and the object “Window” will be converted into “hasBoundaryElement”. The “Has” relationship with the subject “BedRoom” but the object “Furniture” will be converted into “isContaining”. The definition of these relationships has two considerations, one is the consistency with the representation in IFC; the other is an appropriate subdivision to make the check more accurate.
This interpretation has been added at the 3rd paragraph in Section 5.

5. In section 5, optimisation strategies have been introduced and significant performance improvements have been showed. However, the introduction of the query optimisation methods is too strategic and it is hard for other researchers to reproduce the results. Many questions need to be answered:
5.1 What is the query environment for RDF data (e.g. some triple store)? Are OWL reasoners used or just treat data as plain RDF?
Response: RDF data is stored in OWL formatted text and is loaded into memory when it is queried. OWL reasoning machine has not been used.

5.2. “In our method, we cluster the triples by the querying entities, and make the entities with more branches in front of the query.” This needs to be much more elaborated to describe clear algorithms or to give examples.
5.3 In Table 2, model size and all the used query examples should be provided (in appendix for example).
5.4 At least one concrete query example should be presented to show how to optimize the query. It is recommended to use one of the query examples in Table 2.
Response(5.2,5.3,5.4): Yes. The former description of the optimization part is too simple. We have add the detailed the interpretation about the triple clustering, the optimization algorithms and also a concrete query example to show the optimization procedure and the result. The section is reorganized and separated as Section 6. The model size in the Table 2 is provided at the paragraph before Table 2.

6. Format and some minor issues:
6.1 In all the SPARQL query listings, it is recommended that classes start with upper case and properties start with lower case.
Response: Thank you. The SPARQL are generated automatically by our program. As a result, we didn’t adjust the representation manually in the manuscript. We will consider to adjust the program to make the generated SPARQL much easier to read.

6.2 Size of models should be clearly provided. The size of all the RDF data should be provided in triples. For example, in section 5, “The added size of the 8 models are 713,372KB”, is it in Revit format, or in IFC, or in RDF? Please provide IFC model size and transformed RDF model size. Similar thing like The size of the original BIM model(Z15-F011MEP) is 93,596KB” in section 6.1.
Response: Revit and IFC are equivalent to the original BIM model, RDF is the size of the

model after we converted. The model size is the Revit model size, and we have annotated it in the manuscript.

6.3 There are some English issues throughout the paper, often with singular and plural forms. Another check would help.
Response: Thank you for your corrections, we have fixed these problems.

Review #2

In this paper the authors propose a framework for regulatory compliance checking using a structured natural language format. Overall I found the paper very interesting to read - but I have some concerns that should be addressed prior to publication:
- To me the introduction lacked a clear research goal - what does your approach seek to improve upon from other approaches? To me this needs to be in the context of the work already done - not just stating your approach + its performance.
- I was unclear about how the drafting of the SNL is actually performed. The reviewers point to the increased flexibility of this approach - but I am unsure if this is for those drafting the regulations? Who do the authors envisage drafting the regulations? Regulation authors? Has any consultation been performed to see if this is acceptable to them?
Response: The research motivation and also some comparisons with the related work especially on SNL are added in the manuscript at Section 3. The SNL is defined to be used by BIM model inspection experts. It can be used to formalize national regulations, but also to the regulations inside an organization, enterprise, or a project. The SNL has been used by the experienced BIM designers, BIM model inspectors, and they show the acceptance of the SNL language after a simple procedure of learning, especially for their natural language like style.

- Do the authors have any reference to back up their assertion that a BIM is too big to be used in totality - modern tuple stores for IFCOwl etc. are efficient - so would like to see results to back up this claim.
Response: Some BIM model size are hundreds of megabytes, while the query has a lot of conditions to filter, leading to a rule to query may cost several minutes, not to mention the entire rule library; that's why we have adopted a lot of optimization methods to improve query efficiency.

- One point that I do not think is explained well enough in the paper is how the SNL is mapped to the SparlQL i.e. how do you know Bedroom = IfcSpace (with name=Bedroom)? This must be an automated process to make checking feasible at scale so how is it done?
Response: Yes. The procedure was not clearly interpreted at the former manuscript.
The concept mapping from SNL to SPARQL is based on the same configuration file. As to

the different mapping from the concepts in the design codes to a concrete BIM model representation, only the configuration is needed to modify. For example, the mapping defines the concept “Bedroom” in the rule library is defined in the IFC files with the entity “IFCSPACE” and its LongName string contains the substring “Bedroom”.
This interpretation has been added at the 3rd paragraph in Section 5.

- How is the optimization of SPARQL queries done? Automatically – semi automatically? The paper lacked detail on this element.
- I also found myself wanting to see more experimental data. I.e. how much does your optimization reduce queries by? See the speed before optimization and the speed after optimization to me is essential to determine if this approach has any merit.
Response: The optimization of SPARQL queries is automatically. Our optimization method do not reduce the query. For the work of reorganizing the query, it is only by reorganizing the query into a more reasonable way to reduce useless matching and backtracking, thus reducing the query complexity. The former description of the optimization part is too simple. We have add the detailed the interpretation about the optimization algorithms and also a concrete query example to show the optimization procedure and the result. The section is reorganized and separated as Section 6.

Review #3

(1) Originality
Probably, this approach could just as well be done with a regular schema less graph database, which are considered to have user-friendly query languages. So, that probably even defeats the need for an SNL language. Why use semantic web technologies in the suggested approach: what did you gain?
Response: The work of this article is based on OWL and SPARQL. Even with the graph database, SNL language is also necessary, because the contents of the building codes are often complex, writing query sentences directly is difficult and may cause errors. SNL is more convenient and clearer. And also, even with the graph database, we also need to generate a graph database query statement, as in this work to generate SPARQL query statement. So from these two perspectives, using Semantic Web technology is not a worse case than the graph database. At the same time, we are also exploring the use of OWL reasoning technology to complete the information of the model. This function is not available in the graph database, so we chose the semantic web technology.

(2) Significance of the results
I personally do not believe in the possibility of creating such a language (formal and user-friendly are like fire and water), nor in the real need for such a language. End users in the industry, especially AEC, need GUIs, not a query language. User-friendliness depends on the interface, which is not formal anyway. So, a mapping needs to be made from interface to computer language.

Response: For the user, the GUI is really more convenient. We have also tried some GUI work, but unfortunately, GUI is not able to fully meet the needs, especially for complex building specifications. At the natural language level to describe the building codes has been very complicated, if the use of the GUI to describe the rules of the building code, the situation will become more complex. So we designed the SNL language, so that as close as possible in the natural language expression at the same time, with a certain structural and normative. Admittedly, the normative and ease of use of language is difficult to have both, so in the design of SNL we have some consideration, such as the negative form must use "not" to describe, resulting in “not has” expression of this way, which is not in line with the expression of natural language description, but we think that this situation will not give users more distress, it is understandable, so we retain this writing.
At the same time, we also provide users tools to write SNL, which can be regarded as a user-oriented interface. The tool provides a number of SNL commonly used syntax and keywords, when writing rules, users only need to enter very little content to write a correct SNL statement. We believe that the use of SNL in comparison with the GUI, can make the rule entry work more clearly.

(3) Quality of writing
The structure of the paper is great. The style of writing is good. Spelling is okay. The references are poorly formatted. Please use a consistent format, as requested by the journal, to structure references.
Response: This part has been corrected. Thank you for your correcting.

Detailed comments:
----
1. Common query languages and rule languages can be understood by human beings and machines. So why develop yet another language?
Response: For non-computer design domain users, the cost of learning the SPARQL language and learning the SNL is certainly not an order of magnitude. SNL language in the grammar and semantic level are more close to the natural language, such as "bedroom", "has" and other components, attributes and relationships has no difference from the natural language description. The user only need to pay attention to the language structure. And the SPARQL language on these components, attributes and relationships have been completely out of the natural language environment, for the ordinary users, "bedroom" and "?x rdf:type ifc2x3:ifcspace . ?x rdf:LongName ‘bedroom’" which is easier to understand? I think the answer is obvious.

2. You suggest to build rule libraries in the SNL language. That seems quite 'off'. Would it not be a better choice to build libraries of the more formal SPARQL queries? The SNL representation is just an interface to the end user; it can probably be generated from the stored SPARQL queries? In addition, nobody is capable of parsing SNL rules; SPARQL queries can easily be exchanged and run in different systems. Perhaps you can take a look at the recent article in http://www.sciencedirect.com/science/article/pii/S1474034617301945? There are

reasons to choose for a query language and not a rule language, but "able to handle large size models" is not one of them.

Response: As you mentioned, SNL can be seen as a user-oriented "interface". In the whole work, the "rule library" concept actually has two meanings, one is the '.snl' file which only retains the SNL statement, and the other is the '.spl' file which retains both the SNL statement and SPARQL statement. The former can be directly in the SNL editing tools for editing, browsing and preservation, while the latter is actually hidden from the user, used in the follow-up checking process. The conversion from the former to the latter is automatically achieved by the tool. This process is also hidden from the user, so the user can only see the SNL layer. This is why we often refer to SNL when we refer to the concept "rule library". In addition, if the rule library is stored directly as SPARQL, then it cannot be edited and ensure the correctness as easy as the SNL style , which means the ease of use will be greatly reduced.

The recent article you provide about the performance benchmark is really interesting. Thank you. It provides a performance benchmark over semantic rule checking approaches in construction industry, which gives detailed experimental data and suggestions for the selection of ontology reasoners. In this paper, however, instead of focusing on the capacity of ontology tools, we propose a light-weighted method which rule checks big BIM models based domain knowledge on building codes and the feature of BIM models. The literature is referred and we give an interpretation at the 2rd paragraph in Section 1.

3. You distinguish 'declarative' and 'conditional' sentences in the SNL language. The declarative sentence is basically defined as a rule with a universal qualifier ('every' / forAll). What about sentences with existential qualifiers ('there is' / forSome)? Or, well, a bunch of other formal structures available in a formal rule language... Furthermore, the conditional sentences that you mention are typically considered to be part of a declarative language. So, this terminology and classification is odd.
"The SNL supports two kinds of data types: string and digital". I don't know what you mean with 'digital', but I guess that you refer to numbers (integers, floats, and doubles)? What about Booleans, dates, and other data types commonly used everywhere?
NOT is also a logical operator. It might make sense to place this in the list with AND and OR.
Response: "Every" and "There-is" sentences are all simple sentences, because they don't have conditions. Digital means number, we've corrected it in the paper, thank you very much. Since the Boolean type exists as a grammatical element in the natural language, the Boolean type is implemented by the keyword "no" or "not". Other data types (such as date, time, etc.) in the building code has almost no application, so SNL temporarily did not achieve them. NOT in the SNL is similar to the Boolean value, so it is not listed in the logical conjunctions. The logical conjunctions here are more closely related to the concepts in natural language, and are essentially used to connect sentences. NOT is mainly used as a clause rather than a conjunction.

We have modified and supplemented these interpretations at the 5th paragraph in Section 3.

4. How will a machine automatically know how to translate this into a SPARQL rule?
Response: SNL to SPARQL conversion is done automatically by the program. To fulfill the transformation from an SNL to a SPARQL, there are two key mappings, the concept mapping and the relation mapping. First, the concept mapping from SNL to SPARQL is based on a configuration file. For example, the concept of "Bedroom" in the SNL, will be transformed to a SPARQL query as to the entity instances of "IFCSPACE" with their LongName "Bedroom". Second, the relation in an SNL sentence “has” is a more high-level relation description, to make the rule formalization more user-friendly. We map the relation “has” to one of the several relations in the BIM model like "isContaining","hasBoundaryElement", "isConnectedTo","hasSubType",etc. based on different objects and subjects of the "Has" relation. For example, The "Has" relationship with the subject "BedRoom" and the object "Window" will be converted into "hasBoundaryElement". The "Has" relationship with the subject "BedRoom" but the object "Furniture" will be converted into "isContaining". The common sense of the building domain are applied to fulfill the automatic transformation.Numbers don’t have units now, because the OWL model also does not contain units, we can agreed the unit to the same.
This interpretation has been added at the 3rd paragraph in Section 5.

5. If the user is not supported in picking from a list of available terms, the SNL rule can contain anything, making it non-formal, and very expensive to reformat into checkable SPARQL queries (see above).
Response: The SNL rule library can exist independently. The association and the concept mapping between the rule library represented by SNL and the BIM model to be checked is implemented by the configuration. Even if not converted to SPARQL, the SNL sentences are also readable, which makes the validation of the correctness or the rule library easier. It makes sense. The solutions which encode the rules by picking terms from the BIM models like SMC cannot do that.

6. I am missing a reference here to the work around NLP and building regulations, as documented in
http://www.sciencedirect.com/science/article/pii/S0926580516301819.
Response: Thank you. NLP is a great technology, it can help to further automate the whole procedure. Using NLP in our work is what we have been studied and explored. We have achieved some pre-identification and pretreatment of building codes with NLP technology, but this is not part of the main process of our building model check method. However, due to the inaccuracy of the semantics of natural language, many of the Building codes which are written in natural language need further interpretation by domain experts. As a result, it is difficult to translate the codes directly into correct rules by NLP techniques. We still need a good formalization to represent rules of building codes. The SNL does not directly deal with the natural language specification, but the rules of the SNL statement are written

by understanding the building code. Based on the similarity between the SNL and the natural language, the SNL statement is readable for the user and easy to understand.
This part has been added as the 2rd paragraph in Section 3.

7. This is also the entire idea behind the SimpleBIM procedure as documented in https://biblio.ugent.be/publication/8041826 (please read this).
Response: Thank you. The existent research SimpleBIM is really related to the extraction part of our work. It provides several approaches which makes great simplification of the RDF graphs. We appreciated the approaches very much. The ideas like building direct relations between different concepts are very similar with our methods. In this part, our method is not novel. However, there is one main difference between our work and the former simplification research. The goal of our work is regulation checking, thus the extraction procedure is regulation oriented. For example, if the regulation contains the geometry related items to be checked, the geometry information should be extracted, and cannot be simplified by the proposed methods in SimpleBIM. On the other hand, the properties which are irrelevant to the regulation can be deleted safely.

The related work are referred and the interpretation is added in the 2rd paragraph in Section 4.

8. Most, if not all, automated code compliance initiatives using semantic web technologies that rely on SPARQL, take strong advantage of the CONSTRUCT feature to encode the IF-THEN rules. The IF-THEN structure clearly maps well with those CONSTRUCT queries. Similarly, have you considered the use of SPIN?
Response: You are right. In the current check process, the use of SELECT statement can fully meet our needs. In the next step, we will consider using CONSTRUCT and SPIN to handle some complex scenes.

9. This seems to indicate that this step requires both a domain specialist and a programmer. How else can one 'adopt domain knowledge'?
Response: Domain experts only need to learn SNL language to write rules, and do not need to learn programming.

10. It is seamless and automatic as soon as a programmer has implemented the conversion, but he will likely need to do this again and again (so, not that seamless and automatic after all). Please provide more evidence here.
Response: A common program can achieve automatically converting. SNL structure is designed by us, so we also designed a program, parsing the SNL statements, designing and implementing how each part be transferred to SPARQL, and then generate SPARQL query sentences.

11. The tool at http://sts.thss.tsinghua.edu.cn:8079/bimchecker/ was not available when I tested it. I hope that this web page contains datasets and not just a running

program. It would be valuable to see the actual data (e.g. the IFC2X3 ontology used).
Response: We are so sorry for the broken link. We've fixed it.

12. Textual remarks
Response: Thank you very much for your careful work, we have fixed all these mistakes.

Semantic Web 0 (2017) 1–0
IOS Press

Semantic web based rule checking of real-world size BIM models: a pragmatic method

Hehua Zhang [a,*,**], Wenqi Zhao [a], Jianqiao Gu [a], Han Liu [a], and Ming Gu [a]
[a] *School of Software, KLISS, TNLIST, Tsinghua University,Beijing, 100084,China*
E-mail: zhanghehua@tsinghua.edu.cn

Abstract. Rule checking is important to assure the integrity, correctness and usability of Building Information Models (BIMs) in Architecture, Engineering and Construction (AEC) projects. Semantic web based rule checking of BIM models are widely accepted and studied recent years. This technology has noteworthy advantages on interoperability, extensibility and logical basics. However, there are still some gaps to make it practical. One challenge is the efficiency problem on processing large-size BIM models. The other is how to effectively input checking rules which can be understood by both human beings and machines. In this paper, we propose a pragmatic method to check real-world size BIM models. In our framework, BIM models are transformed into a well-defined OWL model. Rules are formalized by a structured natural language (SNL) designed intentionally to describe building regulations. The checking engine is based on SPARQL queries on OWL models. We propose a rule-based model extraction method and optimization strategies on SPARQL statements, which can effectively improve the time efficiency and deal with large-size applications. A prototype has been implemented and applied to BIM models of a real-world building project. We found out non-trivial problems in a totally automatic way, which helped to improve the quality of BIM models and verified the usability of our method.

Keywords: rule checking, BIM, Semantic Web, OWL, SPARQL

1. Introduction

The concept of Building information model (BIMs) has had a tremendous impact on the Architecture, Engineering and Construction (AEC) industries. Its digital and uniform representation of both geometry and all the information related to a building, made it widely accepted and applied in various AEC projects. It is well known that many regulations should be checked and obeyed during the whole life cycle of a building, to make a safe and usable construction product. As the wide use of BIM models, it is important to check the rule compliance of BIM models to assure their integrity, correctness and usability. Since manual checking is time-consuming and error-prone, many researchers have made great efforts on automatic rule checking of BIM models. Typical tools for code checking are Solibri Model Checker (SMC) [1] in Finland, EDM [2] in Norway, ePlanCheck [3]in Singapore and SMARTcodes [4] by ICC (International Code Council) [5], etc.

BIM technologies are further and deeply applied through information fusion with GIS, Infra, etc. to implement smart city applications. In these scenarios, generic and transboundary model representations are more welcome than the building specific representation like the standardization through Industry Foundation Classes (IFC) [6]. Semantic web technologies have great advantages on interoperability, extensibility and logical basics. Consequently, the usage of seman-

*Corresponding author. E-mail: zhanghehua@tsinghua.edu.cn.
**This research is sponsored in part by NSFC Program (No. 61202010, No. 61527812), National Science and Technology Major Project (No. 2016ZX01038101), and the National Key Technology R&D Program (No. 2015BAG14B01-02).

tic web technologies in the domain of AEC are popular and widely studied. Especially, rule checking of BIM models with semantic web technologies gained great attention [7,8,9,10], since they provide originally formal representation of concepts, relations and rules. Representing design codes with rule description languages like SWRL [11], N3Logic [12], and then taking ontology reasoners like Jess [13] for checking are popular solutions. The literature [14] provides a performance benchmark over semantic rule checking approaches in construction industry, which gives detailed experimental data and suggestions for the selection of ontology reasoners. In this paper, instead of focusing on the capacity of ontology tools, we propose a light-weighted method which rule checks big BIM models based domain knowledge on building codes and the feature of BIM models.

With the persistent research achievements on this research topic, however, there are still big gaps to make it practical to solve real-world problems. One challenge is the efficiency problem on processing large-size BIM models. With the continuous application of BIM technology, information becomes increasingly abundant and the size is also growing. In addition, the information in BIM models is not independent. They have various relationships and are intertwined together to form a complex network. Further, a BIM model is usually built as a linked model composed by several BIM models in different disciplines. Therefore, applying semantics web technologies on rule checking real-world BIM models face the difficulties of both time and space cost.

The other challenge is how to effectively input checking rules which can be understood by both human beings and machines. Solihin etc. [15] classified the rules into 4 classes, according to their checking difficulties. How to formalize the rules appropriately is still a question. Logic-based representations are formal and expressive, which are good to machines, while hard to use by human beings. On the other hand, the existing regulations are represented by natural language, which face semantic ambiguity problems and cannot be processed directly by machines. Furthermore, the variety of regulations requests a flexible and extensible representation of rules. Hard-coded or template-based rule representations seldom meet these requests.

In this paper, we propose a pragmatic method for automatic rule checking of BIM models. In our framework, BIM models are transformed into a well-defined OWL [16] model. Rules are described by a structured natural language (SNL) designed intentionally to formalize building regulations. The checking engine is based on SPARQL [17] queries on OWL models, enriched by checking-oriented designs to make it faster. We propose a rule-based model extraction method and several optimization strategies on SPARQL statements, which can effectively improve the time efficiency and deal with large-size applications. A prototype has been implemented and applied to BIM models of the real-world building: Z15 Tower project in China. It found out non-trivial problems in a totally automatic way, which helped to improve the quality of BIM models and verified the usability of our method.

The paper is organized as follows. We introduce the method framework in Section 2. The proposed SNL language is presented in Section 3.The semantic extraction of BIM models are introduced in Section 4. The checking engine and the optimization strategies are presented in Section 5. Section 7 introduces case studies in real-life applications. Finally, we conclude and put our work in perspective in Section 8.

2. Proposed Method

The proposed rule checking method of BIM models based on semantic web technologies is shown in Fig. 1. First, we propose a new SNL language for rule formalization. The building codes are formalized with SNL, so that we get well-defined rule libraries for different checking goals. Second, we propose a semantic model extraction and transformation method, which extracts an appropriate subset of the original BIM models. The subset contains only the necessary information for the targeted rule library. It is then transformed into an OWL model. Finally, all the SNL rules are transformed into SPARQL queries on the OWL model, according to the agreed structure organization. After applying SPARQL optimization strategies, and calling Apache Jena [18] for a single SPARQL query, the checking engine gets the final checking result by summarization and analysis. The rule checking results can be pass, fail or unknown.

The feature of our research framework is to handle real-world size BIM models. Therefore, we take SPARQL queries as the basis of checking engine. We apply light-weighted use of SPARQL queries, while keeping the interpretations, organization, and the control of checking flow by our own checking engine. We can thus design and apply effective and featured optimizations in our framework.

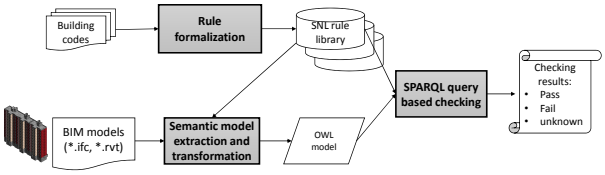

Fig. 1. The architecture of the BIM rule checking method

3. The SNL language

With semantic web technologies, checking rules can be formalized by logic based language like SWRL. However, it is not easy to understand for human beings. Consequently, it cannot be an effective input solution for domain experts. On the other hand, some tools chose hard-coded or template-based rule input method. These methods are less flexible and less extensible, since the rules to be checked cannot be customized easily. In this paper, we propose a structured natural language (SNL) as the rule inputting solution, which is designed intentionally for building codes description.

Converting (controlled) natural languages directly into computable rules with Natural language processing (NLP) techniques are attractive, and there are existing researches on it [19,20]. However, due to the inaccuracy of the semantics of natural language, many of the Building codes which are written in natural language need further interpretation by domain experts, to make its semantics clear and operatable. As a result, it is difficult to translate the codes directly into correct rules by NLP techniques. We still need a good formalization to represent rules of building codes. Therefore, we propose a semi-automatic method. First manually understand the building code, then use SNL language to enter the rules, and build a rule library for further checking. Compared with the existing method such as RASE [19], the SNL does not directly deal with the natural language specification, but the rules of the SNL statement are written by understanding the building code. Based on the similarity between the SNL and the natural language, the SNL statement is readable for the user and easy to understand, the rules will not contain labels and other information independent of the specification itself; and SNL has no need to do re-logical reorganization or other processing.

There are also other existing controlled natural languages like SQUALL [21] and SBVR [22]. The difference of SNL and SQUALL is that, writing SQUALL statement will use the concept of RDF, which means exposing the intermediate data in the model checking process to the user who writes the rules, it is not only difficult for the user to understand but the exposures of such data are not needed. The SNL language is defined in the higher semantic abstraction level. Its writing logic and conceptual description are more similar to the natural language, the user only need to understand the concept of natural language on it, and do not need to understand the knowledge of the Semantic Web. Compared with business rule methods such as SBVR [22], SNL is able to deal with special cases in the building codes which are not easy for business rules description language, such as the reference among different building codes, or variety of conditions in an item, etc.

Compared with the powerful template languages, as an example provided by SMC, the SNL language has the advantage of concise, easy to reuse, and compositional features. Furthermore, since the SMC template language relies on choosing from the BIM models for the concept and relation description, it cannot be used independently as a formal description of building codes. At the beginning of the design, SNL is for the building code, taking the special aspects of the field of construction expertise and domain knowledge into account. In contrast, the SNL language is suitable for this work. It can then be easily validated by the domain experts whether the rule libraries reflect the requests in natural language or not. That is also a crucial problem to make the code checking results trusted.

The SNL language is organized by sentences. There are two basic types of SNL sentences: the simple sentences and the conditional sentences. The simple sentences are identified by the keyword ***Every*** or "There-is-a" to express basic rules with no conditions. The conditional sentences are identified by the keywords ***If*** and ***Then*** to denote which requests should be obeyed when the given conditions are satisfied. Since the building codes is usually a constraint on the properties and relationships of the building components, the SNL is mainly designed for both. The SNL language describes the relationship constraint R of two components a and b with the form $a\,R\,b$. The relationship R of the components is determined by "has", "border-has", "distance between" and so on. The SNL describes the data property constraints P of the component a with the form a ***its*** $P\,op\,Expr$. The data types supported by SNL are mainly composed of strings and numbers. Since the Boolean type exists as a grammatical element in the natural language, the Boolean

type is implemented in the SNL by the keyword ***no*** or ***not***. Other data types (such as date, time, etc.) are not used in the building code we have already concerned, so SNL did not contain them. It can be easily extended in the future if needed. The symbol op denotes the string comparison operators like ***contains***, ***notcontains***, ***equals***, ***notequals***, or numerical comparison operators like $=,\geq,\leq,<,>$. Simple sentence phases are composed together by the logical operator like *and*, *or*. Currently, SNL is able to cover more than 85% of the national building codes (e.g. code for fire protection design of buildings in China (GB 50016-2014): 101/112) and more than 95% of the domain codes.

We present some realistic rules we have used in the applications to gain an intuitive understanding of the SNL language and the rule formalization by it.

1 ***Every*** Bedroom Has Window.
2 ***Every*** LivingRoom ***its*** area $\geq$ 10

The first SNL sentence denotes a request (No. 7.2.1) in the design code for residential buildings in China (GB 50096-2011) that every bedroom should have windows. It is implemented in SNL by the "Has" relation request between the entity Bedroom and the entity Window. The second one denotes another request (No. 5.2.2) in the same code that the area of every livingroom should be bigger or equal to 10 m^2. It is formalized by requesting the value range of the data property "area" of the component "LivingRoom". As can be seen from the two examples, the SNL language is close to the natural language and thus easily understood by human beings. Furthermore, the SNL language still keeps the accurate and strict semantics, so computers can process them too. More complicated and general examples involve the compositions of simple sentence phases, which are shown as follows.

1 ***If*** Building Has Space ***and*** Space ***its*** elevation > 0 ***and*** Space ***not*** Has Window ***and*** Space ***its*** area > 50
Then Space Has ExhaustOutlet.

This SNL sentence denotes a request (No. 8.5.4) in the code for fire protection design of buildings in China (GB 50016-2014). It requests that for all the spaces above ground, if they have no windows and their area is bigger or equal to 50 m^2, they should contain exhaust facilities. The given condition is described in the ***If*** branch and the request is described in the ***Then*** branch. The logical operator ***and*** is applied to connect the simple phases together. This example indicates that the SNL language is powerful to express various complicated sentences with the composition of sentence fragments.

4. Semantic extraction of BIM models

Real-world BIM models are often large in size, which makes the practicality of automatic rule checking a challenge. A building product usually requires a variety of compliance checks. Generally, different specifications relate only to part of the BIM model to be inspected, like the components, their data properties, and the relationships between components. Working on the complete BIM model will result in inefficient procedure. Therefore, we propose a rule library based semantic extraction method for BIM models. It can automatically extract the entities, attributes and relations according to the content of the specification to be inspected.

There are existent research like IfcWoD [23] and SimpleBIM [24] which are closely related to our sub model extraction work. They provide several approaches which make great simplification of the RDF graphs. We appreciated the approaches very much. The ideas like building direct relations between different concepts are very similar with our methods. In this part, our method is not novel. However, there is one main difference between our work and the former simplification work. The goal of our work is regulation checking, thus the extraction procedure is regulation oriented. For example, if the regulation contains the geometry related items, the geometry information should be extracted, and cannot be simplified by the proposed methods in SimpleBIM. On the other hand, the properties which are irrelevant to the regulation can be deleted safely.

The extraction method for an IFC formed BIM model is illustrated in Fig. 2. It contains three core steps. First of all, according to the specification, the directly involved entities and their attributes are extracted as the entity set $E1$ and the attribute set A. By analyzing the SNL statements in the rule library, we can get the components and attributes of the SNL description and apply them to the IFC extraction process. The concepts in the rule are mapped into the components in the IFC through a configuration file (.cfg). For example, the mapping defines that the concept "Bedroom" in the rule library is mapped into the IFC file with the entity "IFCSPACE" and its LongName string contains the substring "Bedroom". As to the different mapping from the concepts in the design codes to a

concrete BIM model representation, only the configuration is needed to modify.

For the rules related to geometric computation (which we can get from the SNL parsing process), we need to extract the information such as bounding box, MEP Points and so on, and obtain the geometry entity set $E2$. By sorting out the existing specifications, we have extracted the commonly used computational keywords, such as "distance", "vertical net distance" and so on. The rules with these keywords must be geometrically calculated to check. By matching these keywords in the process of SNL semantic parsing, we can see which rules need to be geometrically calculated.

The semantic enhancement is to provide more detailed entity semantics than the one in the original IFC file. For example, the IFC file exported by Revit 2015 may represent both pipes and cable carriers as $IFCFLOWSEGMENT$, while we can enrich the semantics by analyzing the related information, and change the entities into $IFCPIPESEGMENT$ and $IFCCABLECARRIERSEGMENT$, respectively. The enriched entity set is denoted by the set $E3$. Therefore, we get the extracted entity set E from the three sources.

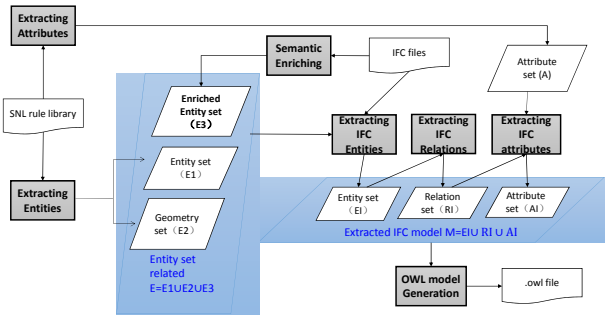

Fig. 2. The semantic extraction and OWL transformation of BIM models

Second, we extract the entity set in the IFC file according to the set E to get the set EI. For any e belonging to EI, we extract the IFC relational element referencing e to get RI. For any r belonging to the RI, we extract the attribute set that belongs to the set A and referencing r (for example $IFCRELDEFINESBYPROPERTIES$) to get AI. Other relationships with types, materials are also considered, if the design codes have the constraints on them. As a result, we get the final IFC sub-model M which is oriented to the rule checking requirements. The comparison on the size of BIM models before and after the extraction is indicated in Table 1. The number of entities and attributes are calculated and compared. Three BIM models on three code libraries are examined. The first model is the MEP model for the 11th floor of the Z15 Tower project (Z15-F011-MEP). The second model is the architecture model for the 2nd unit of a residence project. The last model is the 7th floor underground multi-model (Z15-B007) of the Z15 Tower project, composed by 2 architectural models and 6 MEP models linked together. Three code libraries are the Z15 Tower project code, the code for fire protection design of buildings (GB 50016-2014) and the design code for residential buildings (GB 50096-2011). The number of entities of the extracted semantic model without considering any rule library and the ones orienting the three libraries are denoted as E (total), E(Z15 code), E (FP code), E (R code), respectively. The number of attributes without considering any rule library and with the three libraries are denoted as A (total), A (Z15 code), A (FP code) and A (R code), respectively. From the result, we can see that the rule library based model extraction method can reduce the size of the model, especially when only a small part of the model is related. For example, the residence code checking oriented model extraction of Z15-F011-MEP gets a very small one (with only 440 entities and 20,317 attributes).

Table 1
Size comparisons of the extracted models

Statistical items	Z15-F011-MEP	2building-ARC	Z15-B007
E(total)	7,674	15,513	76,210
A(total)	533,854	934,731	5,516,006
E(Z15 code)	5,531	12,538	42,729
A(Z15 code)	402,364	764,453	3,399,650
E(FP code)	4,322	15,309	39,416
A(FP code)	299,630	929,603	2,997,098
E(R code)	440	15,461	8,893
A(R code)	20,317	933,403	593,375

Finally, the extracted sub-model M is transformed into the OWL model by applying Jena APIs. Considering the checking efficiency, we reorganize the structure of the OWL model, rather than keeping it another representation of IFC files like the work in the literatures [25]. Especially, to make the rule checking more efficient, we largely shorten the query path by building the relations between two entities and the attribute access directly. For example, to represent the connection between a duct fitting and a duct segment in Fig. 3, Fig. 4 indicates the representation comparison from the

original IFC model and our generated OWL model. From a duct fitting to the duct it connects to in the original IFC, it requires three layers of search, while in the generated OWL file only one layer of direct search is enough. Furthermore, through semantic enrichment, we can recognize the component as duct fitting, instead of the abstract semantics like flow fitting, which can be a pipe fitting, or a cable carrier fitting, or a duct fitting, etc.

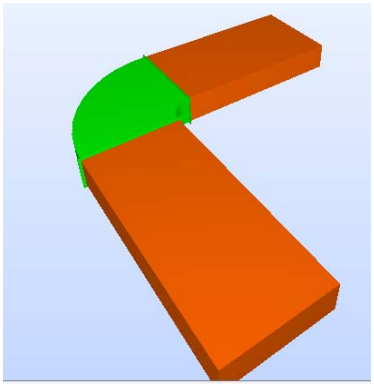

Fig. 3. A fragment model containing a duct segment and its connected duct fitting

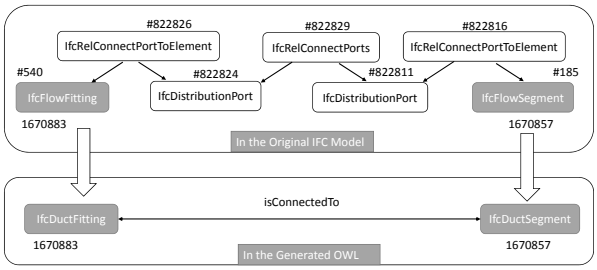

Fig. 4. The representation comparison between the IFC model and our generated OWL model

5. The Checking engine

A building code in a natural language can be described by a series of SNL sentences. We first transform an SNL sentence into one or several SPARQL queries on the extracted OWL model. After that, the checking engine is executed and the result is analyzed. The SNL language is designed to facilitate the description of rules by domain experts, and to facilitate the description of implementation-independent rules. Therefore, the transformation from SNL to SPARQL is an important step to make the rules binding with the BIM semantic model represented in OWL. We can implement the SNL to SPARQL conversion process automatically. SNL statements that conform to the SNL syntax can be converted to SPARQL statements. In general, we propose a reverse query method. To check whether the rule is obeyed, we try to make the SPARQL query to find all the components that violate the rule. If the result set R returned by the query is empty, the checking result is "pass". Otherwise, the rule is violated and we get the related problematic component set C. The SPARQL queries are generated by an SNL syntax-directed style and the transformation is a structural procedure.

We introduce the key points in the transformation by the generated SPARQL queries for the SNL example sentences presented in Section 3 as follows.

SNL: ***Every*** Bedroom Has Window.
SPARQL: SELECT ?X WHERE
?X rdf:type ifc2x3:ifcspace.
?X ifc2x3:LongName ?NOUNVAR0 .
FILTER (regex(?NOUNVAR0, 'Bedroom')) .
FILTER (NOT EXISTS
?Y0 rdf:type ifc2x3:ifcwindow .
?X ifc2x3:hasBoundaryElement ?Y0)

To fulfill the transformation from an SNL to a SPARQL, there are two key mappings, the concept mapping and the relation mapping. First, as we have introduced in Section 4, the concept mapping from a BIM model to the extracted OWL model is based on the configuration file. The concept mapping from SNL to SPARQL is based on the same configuration file. For example, the concept of "Bedroom" in the SNL, will be transformed to a SPARQL query as to the entity instances of "IFCSPACE" with their LongName "Bedroom". Second, the relation in an SNL sentence *has* is a more high-level relation description, to make the rule formalization more user-friendly. We map the relation *has* to one of the several relations in the BIM model like "isContaining","hasBoundaryElement", "isConnectedTo","hasSubType",etc. based on different objects and subjects of the "Has" relation. This is a feature of SNL to be a language to describe design codes. The "has" relationship at the design codes description level, can be converted automatically into different concrete model implementations like the ones in IFC files. We apply the domain knowledge of building to fulfill the automatic transformation. The "has" relationship between the building and the storey is con-

verted to "hasSubType". The other type of "has" relationship can be "isContaining", "hasBoundaryElement", or "isConnectedTo", etc. For example, The "Has" relationship with the subject "BedRoom" and the object "Window" will be converted into "hasBoundaryElement". The "Has" relationship with the subject "BedRoom" but the object "Furniture" will be converted into "isContaining". The definition of these relationships has two considerations, one is the consistency with the representation in IFC; the other is an appropriate subdivision to make the check more accurate.

As a result, the generated SPARQL searches the OWL model and tries to find the spaces which are bedrooms, but without any window on the boundary of it.

SNL: ***Every*** LivingRoom ***its*** area >= 10
SPARQL: SELECT ?X WHERE
?X rdf:type ifc2x3:ifcspace.
?X ifc2x3:LongName ?NOUNVAR0 .
FILTER (regex(?NOUNVAR0, 'LivingRoom'))
.
?X ifc2x3:hasPropertySet ?ps0 .
?ps0 ifc2x3:hasProperty ?p0 .
?p0 ifc2x3:PropName 'area'.
?p0 ifc2x3:PropValue ?v0 .
FILTER(!(?v0 >= 10))

The SPARQL searches the OWL model and tries to find the spaces which are living rooms, but the value of their "area" attributes are not bigger or equal to 10. The more complicated SNL and their transformation to SPARQL are presented as follows.

SNL: ***If*** Building Has Space ***and*** Space ***its*** elevation > 0 ***and*** Space ***not*** Has Window ***and*** Space ***its*** area > 50
Then Space Has ExhaustOutlet.
SPARQL: SELECT ?X WHERE
?X rdf:type ifc2x3:ifcspace.
?X0 rdf:type ifc2x3:ifcbuilding .
FILTER (EXISTS
?X0 ifc2x3:hasSubType ?BS1 .
?BS1 rdf:type ifc2x3:ifcbuildingstorey .
?BS1 ifc2x3:hasSubType ?X) .
?X ifc2x3:hasPropertySet ?ps1 .
?ps1 ifc2x3:hasProperty ?p1 .
?p1 ifc2x3:PropName 'elevation'.
?p1 ifc2x3:PropValue ?v1 .
FILTER(?v1 > 0) .
FILTER (NOT EXISTS
?Y2 rdf:type ifc2x3:ifcwindow .
?X ifc2x3:isContaining ?Y2) .
?X ifc2x3:hasPropertySet ?ps3 .
?ps3 ifc2x3:hasProperty ?p3 .
?p3 ifc2x3:PropName 'area'.
?p3 ifc2x3:PropValue ?v3 .
FILTER(?v3 > 50) .
FILTER (NOT EXISTS
?Y4 rdf:type ifc2x3:ifcairterminal .
?Y4 ifc2x3:Name ?NOUNVAR0 .
FILTER (regex(?NOUNVAR0, 'Exhaust outlet')) .
?X ifc2x3:isContaining ?Y4)

From the examples, we can also verify that the SNL is more concise and easier to use as a rule inputting interface. Through the seamless and automatic transformation, we obtain both the advantages on the rule inputting and automatic rule checking.

6. The optimization strategies

The generated SPARQL queries work well in terms of functionality. However, some SPARQL queries cost much searching time in our practice. Jena cannot do more even with some optimization strategies, since it is a universal tool. However, BIM models have their own features on the entities, attributes and their relations. To process real-world size BIM models, we made BIM domain specific optimization strategies, which brought great improvements on the searching efficiency. The first strategy is the order rearrangement. A SPARQL sentence is generally executed according to the sequence of the triples. That is to say, the order of the triples affects the execution time to a great extent. In our method, we cluster the triples by the querying entities, and place the entities with more branches first in the query. This strategy can make the query quickly converge, avoiding the unnecessary Cartesian product of big sets.

The core of the reorganization of SPARQL queries is the reorganization of the internal structure of the queries, through which, the queries space can be quickly converged when the queries is executed, so as to avoid unnecessary matching and recursive waste of queries time. The specific steps are divided into "forest construction and sequencing" and "triples reorganization".

The first step in the "forest construction and sequencing" stage is determining the tree roots in the forest. All entities concerned in the BIM model will be

selected as roots. After selecting the roots, we traverse all triples in turn. In this step, the subjects, predicates and objects in the triples will be transformed into the parent nodes, edges and child nodes of the tree, respectively. The upper side of Fig. 5 shows an example of converting triples into a tree. From line 1, we turn entity $?X0$ into a root, and we add an edge represent the $rdf : type$ side and the edge connects to a sub-node $ifc2x3 : ifcbuilding$. As the same as to the triple in line 2, we add a child node of $?ps0$ to the root. As to the third line, we add another edge and child node to the $?ps0$ node, and so on. Eventually the tree is formed shown in Fig.5.After adding all the triples to the forest, we sort the trees in the forest by the branch number of the roots. This arrangement will put more constraints on the front of the entity.

```
1 ?X0  rdf:type ifc2x3:ifcbuilding .
2 ?X0 ifc2x3:hasPropertySet  ?ps0  .
3 ?ps0 ifc2x3:hasProperty ?p0  .
4 ?p0 ifc2x3:PropName 'BUILDINGNAME'.
5 ?p0  ifc2x3:PropValue  ?v0  .
```

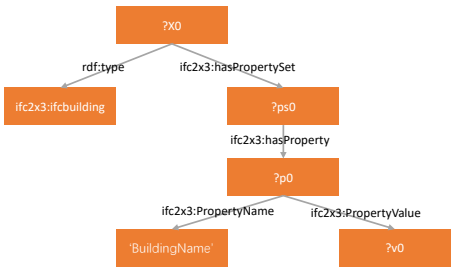

Fig. 5. The tree example of the triples

After "forest construction and sorting", we execute the " triples reorganization " procedure. The purpose of is to convert the triples in the forest into valid SPARQL queries again without changing the semantics of the original queries. This work is implemented by traversing the trees in the forest from head to tail, and make the Depth First Searching (DFS) of each tree. The triples are reorganized while traversing. Algorithm 1 describes this process. The input of the algorithm is the forest generated by the previous introduction. The output are the *triplesOF* strings which generated by traversing the forest. The first line of the algorithm assigns an empty string to *triplesOF* for initialization. Lines 2 to 5 are a loop, and the trees in the forest are traversed in the order of arrangement, with the DFS of each tree in the forest. Line 3 takes the currently traversed tree and line 4 get its root node. The 5th line of the algorithm is to add the result of the triple which is generated by the DFS of each tree to *triplesOF*. The tree traversal algorithm is introduced by algorithm 2. Finally, on line 6, the result of all triples reorganizations is returned.

Input : $forest$ is the orderes forest
OutPut: *triplesOF* is the triples except the filter parts which is recognized from the forest

triplesOF ← EMPTY_STRING
for $i < size(forest)$ **do**
  tree ← $\texttt{getTree}(forest, i)$
  rootNode ← $\texttt{getRoot}(tree)$
  triplesOF ←
  triplesOF + TRANVERSE(*rootNode*)
end
return *triplesOF*

Algorithm 1: COMPSETRIPLES

Algorithm 2 is a recursive algorithm for DFS of trees. The input is a node of the tree, and the algorithm will start traversing from this node. The output triples are a series of triple strings that are combined in the entire traversal process starting from the input node. The first line of the algorithm assigns all child nodes of the input node to the *childList*. The second line initializes triples with an empty string. Lines 3 to 6 are a loop that traverses all child nodes of the input node. Line 4 takes the currently traversed child node assigned to *childNode*. Line 5 began the work of the reorganization of the triples with turning node name into the triple subject, turning the edge to its *childNode* to a predicate, and turning *childNode* name into an object. In line 5, $SPACE$ indicates a space, and DOT means ".". The line completed a reorganization of a triple. Line 6 starts with the currently traversed *childNode*, applies the $Tranverse$ algorithm again, and adds the traversal result to the triples. After the loop, we get the triple strings of the nodes which are traversed in the DFS of the input node.It is also the triple strings of the tree.

After the triples except the filter parts are obtained by the above algorithm, we can get the SPARQL queries by simply splicing the rest of the queries.

The second strategy is the refactoring of the representation. One query can be represented in different forms, with the same functionality, but with different time costs. Taking this into account, we replace some structures by the same functional part, but with better

Input : *node* is a node in the tree
Output: *triples* is the triples which generated by DFS og the node

childList ← `getChildList`(*node*)
triples ← EMPTY_STRING
for *i* < `size`(*childList*) **do**
  childNode ← `get`(*childList*, *i*)
  triples ←
  triples + `getNodeName`(*node*) + SPACE + `getEdgeName`(*childNode*) + SPACE + `getNodeName`(*childNode*) + DOT
  triples ← *triples* + TRANVERSE(*childNode*)
end
return *triples*

Algorithm 2: TRANVERSE

efficiency. For example, the form "FILTER (EXISTS ?Y20 ifc2x3:isContaining ?X)" can be optimized as "?Y20 ifc2x3:isContaining ?X", if it does not occur in another FILTER environment. It can first avoid this part to be executed since the FILTER parts are executed last in Jena. The other advantage is that it becomes a triple so the first optimization strategy can be applied. Finally, we did some pre-queries for some complicated SPARQL statements, and only when the searching result set of the pre-queries is not empty, the real SPARQL query is executed. This strategy saves a lot of query time especially for the multi-level embedded Filter parts, where the first two strategies do not work.

The BIM model of the seventh floor underground (Z15_B007) in the Z15 Tower project is taken to examine the optimization results. It is a linked multi-model which is composed by 8 single Revit models with 2 architecture models, and 6 MEP models. The added size of the 8 models are $713,372KB$ (with the Revit format). We apply our SNL rule library for fire protection (GB50016-2014) on this BIM model, and the execution time for some bottleneck SPARQL queries before and after applying the three strategies are demonstrated in Table 2. The bar in the table represents that the query result cannot be given after 10 minutes. From the table, the optimization strategies improve the performance of key queries to a great extent, and do improve the performance of query based BIM rule checking.

The following shows an example of a complete reorganization of the SPARQL queries. The natural language of of the item 6.9.6 from the design code for residential buildings (GB 50096-2011) and the SNL rule is shown as follows. The generated SPARQL before optimization is shown in Fig. 7. The forest generated during the optimization procedure is shown in Fig. 6. The optimized SPARQL sentence is shown in Fig. 8.

Table 2
Time comparisons of the SPARQL optimizations

Items	time without opt (ms)	time with opt (ms)
5.4.10-1	-	271
5.4.11-6	-	60
5.4.13-6	-	58
5.5.21-30	-	17
5.5.21-28	67838	29
5.5.21-29	24367	48
6.2.2-1	47826	76
6.2.2-5	58387	43
7.3.1-4	11688	20
7.3.1-5	11638	19

We apply both the generated query and the optimized query to the "cell unit building" project (the model size is $38.73MB$, Json format). The query time with the optimized SPARQL query is about $18ms$. On the other hand, the query before optimization applied to the same model takes up to 13580ms. This forest-based reorganization strategy applied to the SPARQL at that time reached an astonishing 99.87% efficiency optimization.

Design Code for Residential Building 6.9.6: The Entrance of the underground floor and elevator which directly connect residential unit should be set up B fire doors. And it is forbidden to use staircases or elevators for the underground garage for natural ventilation.

SNL: ***If*** Elevator Has Door ***and*** Elevator ***its*** elevation < 0 ***Then*** Door ***its*** Fire-proof equals "B"

7. Applications

Based on the proposed method, we developed a prototype tool BimChecker, which can process both IFC models and Revit models. The core of the BimChecker tool is implemented in Java 8. We provide a Web-based version and a Revit Plugin, for different kinds of applications. The Web-based version is implemented based on Apache Tomcat 9. It supports uploading the IFC model (.ifc) and the rule library which has been generated to SPARQL (.xml files). With the selection

```
SELECT ?X WHERE {
    ?X rdf:type ifc2x3:ifcdoor.
    ?X0  rdf:type ifc2x3:ifcspace.
    ?X0 ifc2x3:LongName ?NOUNVAR0 .
    FILTER (regex( ?NOUNVAR0, 'Elevator')
    ) .
    FILTER ( EXISTS {
     ?X0 ifc2x3:hasBoundaryElement ?X }
)  .
     ?X0 ifc2x3:hasPropertySet  ?ps1  .
     ?ps1 ifc2x3:hasProperty ?p1.
     ?p1 ifc2x3:PropName 'elevation'.
     ?p1  ifc2x3:PropValue  ?v1.
      FILTER( ?v1 &gt; 0.0 ) .
      ?X ifc2x3:hasPropertySet  ?ps2  .
    ?ps2 ifc2x3:hasProperty ?p2  .
    ?p2 ifc2x3:PropName 'Fire-proof'.
    ?p2  ifc2x3:PropValue  ?v2  .
    FILTER(!( xsd:string( ?v2 )  ='B' ))
```

Fig. 6. The query generated directly from SNL to SPARQL

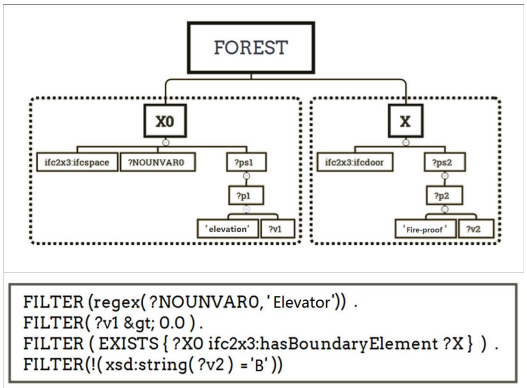

Fig. 7. The forest generated during the SPARQL optimization procedure

of rules in a rule library, it executes the BimChecker core checking engine, and presents the checking result in a webpage [26]. The Revit plugin version of BimChecker can support better interactivity. The interface and the model information acquisition part are implemented in C#, based on Revit APIs, and the checking core is called to implement the checking functionalities. The checking result interface of BimChecker is shown by example in Fig. 9.

The Z15 Tower, located in Beijing CBD core area Z15 plots, is the tallest landmark building in Beijing. The total building height is 528 meters, with 108 floors above ground, and 7 floors underground. The project

```
SELECT ?X WHERE {
    ?X0 rdf:type ifc2x3:ifcspace.
    ?X0 ifc2x3:LongName ?NOUNVAR0 .
    ?X0 ifc2x3:hasPropertySet  ?ps1  .
    ?ps1 ifc2x3:hasProperty ?p1.
    ?p1 ifc2x3:PropName 'elevation'.
    ?p1  ifc2x3:PropValue  ?v1.
    ?X rdf:type ifc2x3:ifcdoor.
    ?X ifc2x3:hasPropertySet  ?ps2  .
    ?ps2 ifc2x3:hasProperty ?p2  .
    ?p2 ifc2x3:PropName 'Fire-proof'.
    ?p2  ifc2x3:PropValue  ?v2  .
    FILTER (regex( ?NOUNVAR0, 'Elevator')
    ) .
    FILTER ( EXISTS {
     ?X0 ifc2x3:hasBoundaryElement ?X }
)  .
    FILTER( ?v1 &gt; 0.0 ) .
    FILTER(!( xsd:string( ?v2 )='B'))}
```

Fig. 8. The generated SPARQL query after optimization

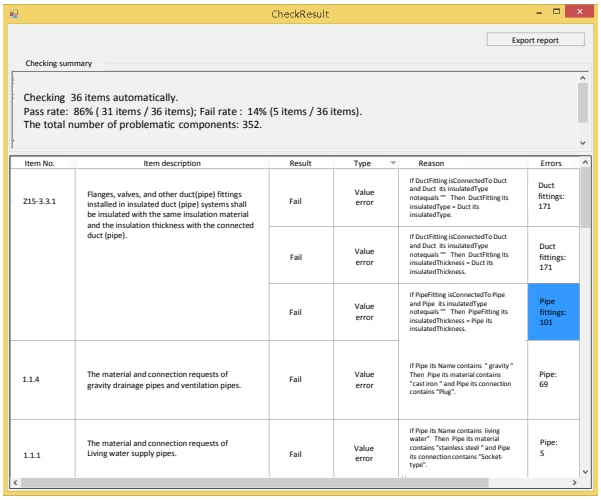

Fig. 9. The checking result shown by the BimChecker tool

started on July 29, 2013, and planned to complete in October 2018. As a typical super-high-rise building, the Z15 project faces many challenges in both design and construction phase. The project has strict requirements on application of BIM technologies and the correctness of BIM models. We collaborated with the Z15 project group and applied our BimChecker tool to the electromechanical deep design and multi-model synthesis stages. Several non-trivial problems are found, which are hard to find manually or time-consuming.

The applications on Z15 Tower project verified the usability of our method.

Specifically, we built 5 rule libraries to handle different kinds of checking requirements. The first one aims to check the information correctness of drainage component relative to the Z15 project requirements. The second one aims to check the supports and hangers related national standards compliance. The third one checks whether the installation space and maintenance space are enough reserved. The fourth one is for component naming specification conformance. The last one is to check the reasonability for modular cut of pipes.

We illustrate some of the checking results. The first rule library contains 36 items which cover the property information correctness checking of all kinds of pipes, pipe fittings,etc., with specific conditions. The 36 natural language described items are formalized as 116 rules with our SNL language. The tool then generates 116 compliance SPARQL queries. The MEP BIM model for the 11*th* floor in Z15 project as shown in Fig. 10 was examined. We found that 5 items failed on compliance checking,with 352 problematic components. The checking result interface is shown in Fig. 9.

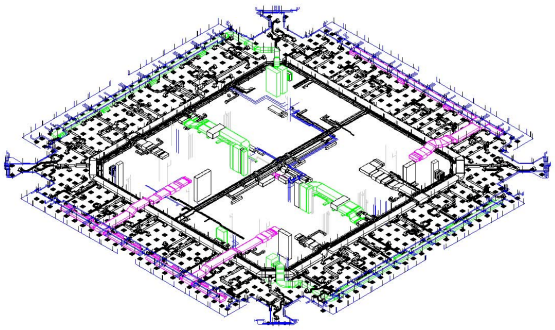

Fig. 10. The MEP BIM model of the 11th floor in the Z15 Tower project

To illustrate, one item (No. Z15-3.3.1) among the 5 failed ones requests flanges, valves, and other duct (pipe) fittings installed in insulated duct (piping) systems shall be insulated with the same insulation type and the insulation thickness with the connected duct (pipe). It is formalized in our tool by 4 SNL rules. One rule respective to pipe fittings and their insulation thicknesses are shown as follows.

SNL: ***If*** PipeFitting isConnectedTo Pipe ***and*** Pipe ***its*** insulatedType ***notequals*** ""
Then PipeFitting ***its*** insulatedThickness = Pipe ***its*** insulatedThickness.

The tool found that 101 pipe fittings are non-compliant with this rule. The problematic components are well demonstrated and located by our BimChecker tool, so that the modeler can quickly find them and fix them. For example, as shown in Fig. 11, one problematic component found by our tool is identified as *"Z15_CCIEI_Z01_HVAC_Reducing tee fittings HVAC _Chilled water backwater 1554291"*. Its insulation layer type is "flexible foam rubber" and its insulation layer thickness is 35.0mm. It is connected to a pipe identified as *"Z15_CCIEI_Z01_HVAC_Chilled water backwater 1554289"*, with the insulation layer type "flexible foam rubber", and the insulation thickness of 30.0mm. The insulation thickness is not the same value as the rule requested.

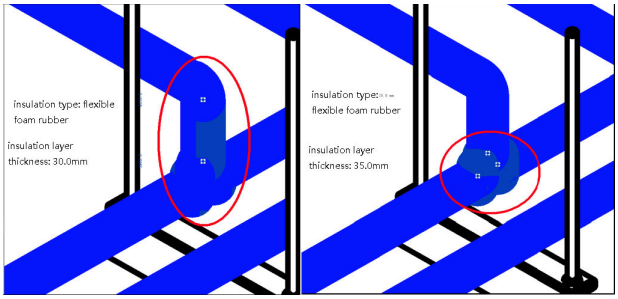

Fig. 11. An instance of the problematic tees

The size of the original BIM model (Z15-F011-MEP) is 93,596KB (with the Revit format).The number of entities of the extracted OWL model is 5,531, while the number of the attributes is 402,364, as shown in Table 1. The whole execution time is 21.98 seconds,including the procedure of model extraction (11.73s), model transformation (5.58s), the SPARQL queries (2.97s), and the checking result generation (1.7s). The average execution of a SPARQL query is 25.6 ms. It validates that our semantic web based rule checking method can be effectively used to checking BIM models in practice.

8. Conclusion

Rule checking is important to assure the integrity, correctness and usability of Building Information Models (BIMs) in Architecture, Engineering and Construction (AEC) projects. Semantic web technologies based rule checking of BIM models are widely ac-

cepted and studied in recent years. However, the difficulties on processing large-size real-world BIM models and on effective inputting checking rules are not totally solved. In this paper, we propose a novel method to automatic rule checking of BIM models. First, we propose the SNL language for effective rule input by domain experts (rather than programmers). The SNL rules are easily read and understood by human beings, thus made the rule library validation possible. Second, a semantic model extraction and transformation method is proposed. The rule library oriented method helped to reduce the size of the model and improve the efficiency of checking. Finally, the OWL model checking based on SPARQL query is proposed, which can support rule compliance checking. By optimizing the checking flow, checking results of real-world BIM models can be given effectively and fast. We developed a prototype tool BimChecker and applied it to the Z15 Tower project. Many significant problems were found, which helped to improve the accuracy of the model. The applications also validates the usability of our methods.

In the future, we plan to build more rule libraries, and apply the suggested approach to other practical buildings like LianTang Port in Shenzhen, China. We also plan to design more domain specific optimization strategies to further reduce the checking time, especially the time of model extraction and transformation.

References

[1] SMC:the Solibri model checker, http://www.solibri.com.

[2] EDM model checker, http://www.epmtech.jotne.com/index.php?id=512200.

[3] novaCITYNETS. implementing IFC-automatic code checking(e-plancheck), http://www.nova-hub.com/e-government/.

[4] D. Conover. Development and implementation of automated code compliance checking in the U.S. *International Code Council*, 2007.

[5] ICC: International code council, https://www.iccsafe.org/.

[6] BuildingSMART international, summary of IFC releases, available online: http://www.buildingsmart-tech.org/ifc/ifc4/final/html/index.htm.

[7] Y.-s. Jeong J.-K. Lee C. M. Eastman, J.-m. Lee. Automatic rule-based checking of building designs. *Automation in Construction*, 18 (8):1011–1033, 2009.

[8] R. Verstraeten-J. De Roo-R. DeMeyer R. Van de Walle J. Van Campenhout P. Pauwels, D. Van Deursen. A semantic rule checking environment for building performance checking. *Automation in Construction*, 20 (5):506–518, 2011.

[9] S. Zhang P. Pauwels. Semantic rule-checking for regulation compliance checking: An overview of strategies and approaches. In *Proceedings of the 32rd International CIB W78 Conference*, pages 619–628, 2015.

[10] H.Li T.Kasim T.H.Beach, Y.Rezgui. A rule-based semantic approach for automated regulatory compliance in the construction sector. *Expert Systems with Applications*, 42 (12):5219–5231, 2015.

[11] H. Boley et al I. Horrocks, P. F. Patel-Schneider. SWRL: A semantic web rule language combining owl and ruleml. *W3C Member Submission*, May 2004.

[12] Lalana Kagal Yosi Scharf Tim Berners-Lee, Dan Connolly. N3logic: A logical framework for the world wide web. *Theory & Practice of Logic Programming*, 8 (3):249–269, 2007.

[13] Jess: the rule engine for the java platform, http://www.jessrules.com/jess/index.shtml.

[14] Pieter Pauwels, Tarcisio Mendes de Farias, Chi Zhang, Ana Roxin, Jakob Beetz, Jos De Roo, and Christophe Nicolle. A performance benchmark over semantic rule checking approaches in construction industry. *Advanced Engineering Informatics*, 33:68 – 88, 2017.

[15] C.Eastman W.Solihin. Classification of rules for automated bim rule checking development. *Automation in Construction*, 53:69–82, 2015.

[16] Frank Van Harmelen Deborah L McGuinness. Owl web ontology language overview. *W3C recommendation*, 10 (10), 2004.

[17] SPARQL 1.1 query language. w3c recommendation 21 march 2013, http://www.w3.org/tr/sparql11-query/.

[18] Apache Jena: A free and open source java framework for building semantic web and linked data applications. http://jena.apache.org/.

[19] Nisbet N Hjelseth E. Capturing normative constraints by use of the semantic mark-up RASE methodology. In *Proceedings of CIB W78-W102 Conference*, pages 1–10, 2011.

[20] Nora M. El-Gohary Jiansong Zhang. Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking. *Automation in Construction*, 73:45–57, 2017.

[21] FerrÍę S. SQUALL. a controlled natural language as expressive as SPARQL 1.1. In *Proceedings of International Conference on Application of Natural Language to Information Systems*, pages 114–125, 2013.

[22] Bordbar B Bajwa I S, Lee M G. SBVR business rules generation from natural language specification. In *AAAI spring symposium: AI for business agility*, pages 2–8.

[23] Nicolle C. Mendes de Farias T., Roxin A. IfcWoD, semantically adapting IFC model relations into OWL properties. In *Proceedings of the 32nd CIB W78 Conference on Information Technology in Construction*, pages 175–185, 2015.

[24] Ana Roxin Pieter Pauwels UGent. IfcWoD, semantically adapting IFC model relations into OWL properties. In *11th European Conference on Product and Process Modelling*, pages 11–18, 2016.

[25] W. Terkaj P. Pauwels. Express to owl for construction industry: Towards a recommendable and usable ifcowl ontology. *Automation in Construction*, 63:100–133, 2016.

[26] BimChecker. available at: http://sts.thss.tsinghua.edu.cn:8079/bimchecker/.