Semantic Web 0 (2017) 1–0
IOS Press

Semantic web based rule checking of real-world scale 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-scale 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 scale 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-scale 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 representation 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.

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-scale BIM models. With the continuous application of BIM technology, information becomes increasingly abundant and the scale is also growing. In addition, the information in BIM models is not independent. They have various relationships and 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. [14] 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 existent regulations are represented by natural language, which face semantic ambiguity problems and can not be processed directly by machines. Furthermore, the variety of regulations request a flexible and extensible representation of rules. Hard-coded or template-based rule representations are hard to 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 [15] model. Rules are described by a structured natural language (SNL) designed intentionally to formalize building regulations. The checking engine is based on SPARQL [16] 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-scale 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 6 introduces case studies in real-life applications. Finally, we conclude and put our work in perspective in Section 7.

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 [17] 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.

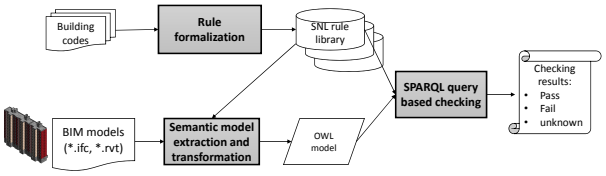

Fig. 1. The architecture of the BIM rule checking method

The feature of our research framework is to handle real-world scale BIM models. Therefore, we take SPARQL queries as the basis of checking engine, rather than reasoning engines. Although taking existent reasoning engine is an off-the-shelf solution, it can only process BIM models in very limited scale, by our

experience. 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.

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 understood by 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 failed on the extensibility or flexibility since the rules to be checked can not be customized easily. In this paper, we propose a novel structured natural language (SNL) as the rule inputting solution, which is designed intentionally for building codes description.

The SNL languages is organized by sentences. It supports two kinds of statements: the declarative sentences and the conditional sentences. The declarative sentences are identified by the keyword ***Every*** to express the rules should be obeyed in any case. The conditional sentences are identified by the keywords ***If*** and ***Then*** to denote the requests should be obeyed when the given conditions are satisfied. The SNL language describes the relationship constraint R of two components a and b with the form $a\,R\,b$. It describes the data property constraints P of the component a with the form a ***its*** $P\,op\,Expr$. The SNL supports two kinds of data types: string and digital. The symbol op denotes the string comparison operators like ***contains***, ***notcontains***, ***equals***, ***notequals***, or digital comparison operators like $=,\geq,\leq,<,>$. Simple sentence phases are composed together by the logical operator like *and*, *or*.

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 living-room should be bigger or equals to 10 m^2. It is formalized by requesting the value range of the data property "area" of the entity LivingRoom. As can be seen from the two examples, the SNL language is close to the natural language and thus easy to be 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 sentences 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.

Compared with the hard-coded solutions, SNL can provide more flexible and powerful expressiveness. 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 can not be used independently as a formal description of building codes. 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.

4. Semantic extraction of BIM models

The BIM models in the real-case are often large in scale, 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 only related entities, attributes and relations according to the content of the specification to be inspected.

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. For the rules related to geometric computation, we need to extract the information such as bounding box, MEP Points and so on, and obtain the geometry entity set $E2$. 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.

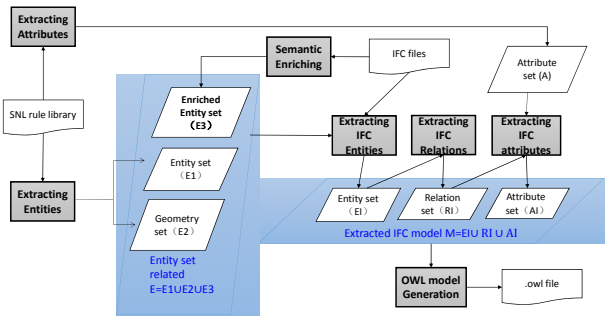

Fig. 2. The semantic extraction and OWL transformation of BIM models

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. As a result, we get the final IFC sub-model M which is oriented to the rule checking requirements. The comparison on the scale 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 modes 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 2th 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 scale of the model, especially when only a small part of the model are related. For example, the residence code checking oriented model extraction of Z15-F011-MEP get 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 [18]. 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 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.

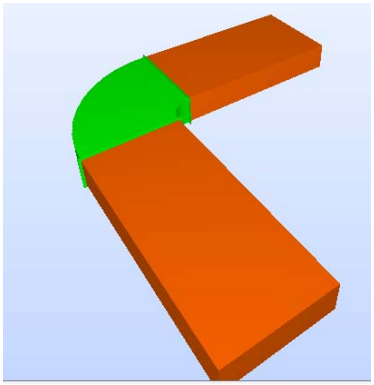

Fig. 3. A fragment model containing a duct segment and its connected duct fitting

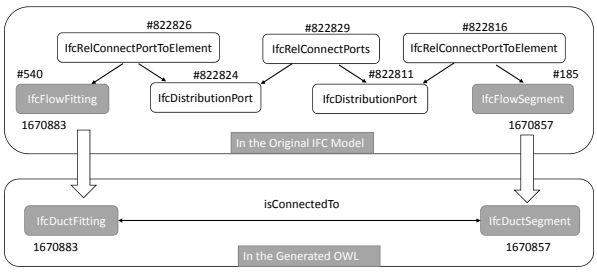

Fig. 4. The representation comparison between the IFC model and our generated OWL model

5. Checking engine and optimization strategies

A building code in a natural language can be described by a series of SNL sentences. We first transform a 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 user description, and implementation-independent. Therefore, the transformation from SNL to SPARQL is an important step to make the rules binding with the BIM semantic model represented in OWL. 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 components set C.

The SPARQL queries are generated by a SNL syntax-directed style and the transformation is a structural procedure. We present 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)

The SPARQL searches the OWL model and tries to find the spaces which are bedrooms, but there does not exist any window which is on the boundary of it. To fulfill the transformation from a SNL to a SPARQL, there are two key mappings, the concept and the relation. We made the mappings automatically by adopting related domain knowledge. For example, the concept "Bedroom" in the SNL sentence corresponds to the $IFCSPACE$ entity with its LongName containing the word "Bedroom". Second, the relation in a 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. For example, through the domain knowledge, we know that the relation between a space and a window should be *hasBoundaryElement*, which denotes that a space has a window on its boundary.

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.

The generated SPARQL queries can work well in terms of functionality. However, some SPARQL queries cost much searching time by our practice. Jena can not 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 scale 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 appearance of the triples affect the execution time to a great extent. In our method, we cluster the triples by the querying entities, and make the entities with more branches in front of the query. This strategy can make the query quickly converge, avoiding the unnecessary Cartesian product of big sets.

The second strategy is the refactoring of the representation. One query can be represented by different forms, with the same functionality, while different time cost. Taking this into account, we replace some structures by the same functional part, but with better efficiency. For example, The form like "FILTER (EXISTS ?Y20 ifc2x3:isContaining ?X)" is optimized as "?Y20 ifc2x3:isContaining ?X", if it does not occur in another FILTER environment. It can first avoid this part to be executed very late since the FILTER parts are executed at 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 $713,372KB$. 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 can not 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.

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

6. 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 by 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 of rules in a rule library, it executes the BimChecker core checking engine, and presents the checking result in a webpage [19]. The Revit plugin version of BimChecker can support better interactivity. The interface and the model information acquisition part are implemented by 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. 5.

6.1. The Z15 Tower applications

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 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,

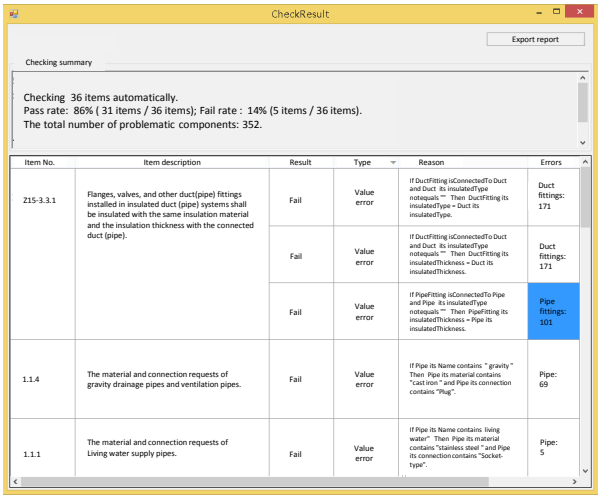

Fig. 5. The checking result shown by the BimChecker tool

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 illustrates some of the checking results for examples. The first rule library contains 36 items which covers 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 $11th$ floor in $Z15$ project as shown in Fig. 6 was examined. We found 5 items are failed on compliance checking,with 352 problematic components. The checking result interface is shown in Fig. 5.

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.

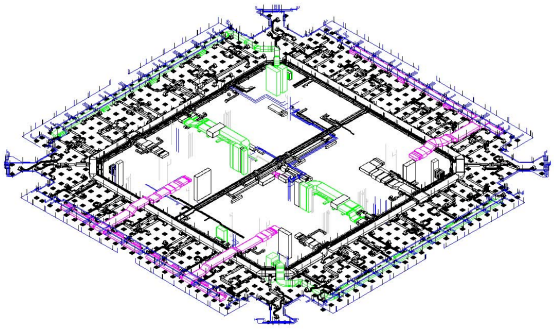

Fig. 6. The MEP BIM model of the 11th floor in the Z15 Tower project

One rule respective to pipe fittings and their insulation thickness are shown as follows.

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

The tool found 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. 7, 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 the 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.

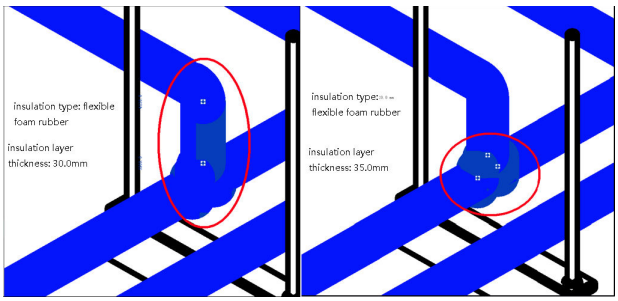

Fig. 7. An instance of the problematic tees

The size of the original BIM model(Z15-F011-MEP) is 93,596KB.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.

7. 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 accepted and studied recent years. However, the difficulties on processing large-scale real-world BIM models and on effective inputting checking rules are still left. In this paper, we propose a novel method to automatically 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 easy to 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 in effective time. We developed a prototype tool BimChecker and applied it into 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 applied to other practical on-built 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] C.Eastman W.Solihin. Classification of rules for automated bim rule checking development. *Automation in Construction*, 53:69–82, 2015.

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

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

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

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

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