Rule Checking Method-centered Approach to Represent Building Permit Requirements

Seokyung Park[a], Hyunsoo Lee[a], Sangik Lee[a], Jaeyoung Shin[a] and Jin-Kook Lee[a]

[a]Dept of Interior Architecture Design, Hanyang University, Seoul, Republic of Korea
E-mail: seokyung.park529@gmail.com, hyunsoolee120@gmail.com, kignaseel@gmail.com, jjyoung311@gmail.com, designit@hanyang.ac.kr

ABSTRACT

This paper aims to describe rule checking method, classification and its demonstration. As applications of BIM extends, there have been some challenging projects on automated building compliance checking. The current rule-making method is developer-centered and thus is difficult to define rules without propound programming knowledge. This paper introduces high level rule making methods with law sentence-centered approach. The proposed methods have intuitive naming convention and are directly mapped with the predicate of the law sentences. Therefore, it is easy to infer function of the methods. According to the type of object and property in instance level, three hierarchies of method classification were set: 1) level 1 divides types of instance, 2) level 2 classifies the type of property, and 3) level 3 specifies the content of checking. From the level 3, representative rule checking method is defined. The representative method is subdivided into extended methods according to the specific object and property to check. The rule checking methods are combined together to form an intermediate pseudo-code. The pseudo-code is later to be parsed into computer executable form. This paper mainly focuses on 1) introducing law sentence -centered rule checking method, 2) object and property-based classification of rule checking method, 3) method extensibility and 4) demonstration of rule checking methods with actual requirement sentences from the Korea Building Permit. The high level rule checking method is developed as a part of KBimLogic. KBimLogic is a software that translates the Korea Building Permit requirement into computer executable format. KBimLogic is now under development with government funding.

Keywords –

Rule Checking Methods; Rule Checking Method Classification; Automated Building Compliance Checking; BIM (Building Information Modeling); Korea Building Code

1 Introduction

As one of the Building Information Modelling (BIM) applications, automated design evaluation has become available in building design process [1]. Especially, code compliance checking, conventionally done manually, benefits from automated design evaluation [2, 3, 4]. There have been some challenging projects in code compliance checking mostly led by governments such as the USA, Norway, Singapore and Australia [5, 6, 7, 8]. In the process of rule checking, rule interpretation, a step that translates requirement written in natural language into machine executable format comes first [9, 10]. One of the important lessons learned from the previous researches is that logical structure of the law sentence is significant in rule representation. Logical structure enables ambiguous human readable requirement to be explicitly executable in computer. As a part of logical structure, we developed rule checking methods. Different from existing rule-making methods, it is law sentence - centered rule checking method.

This paper mainly focuses on introducing the rule checking method and its object's property based classification. As an example structure of rule checking methods, building permit requirements in Korea Building Code were selected.

2 Research Scope and Objective

For the development of rule checking method, four representative articles from Korea Building Code were chosen. They were selected according to the feasibility of application of the logical structure. The selected articles are as follows: 1) Building Act 49, egresses from buildings and restrictions on their use, 2) Building Act 53, regulations on basement level, 3) Building Act 56, construction of double walls and connecting corridors, and 4) Building Act 64, regulation on installation of elevator. Because the Korea Building Code has intricate taxonomic relations, 44 related articles were additionally

chosen.

The rule checking method is developed as a part of the logical structure. The logical structure consists of three modules: object·property module, predicate module, and relation module. The Object·property module handles noun phrase in law sentences while the predicate module treats verbal phrase. The Relation module is in charge of combination of each module and taxonomic relation of sentences. Among the three modules, the rule checking method belongs to the predicate module. It represents verbal phrase of law sentences. The figure 1 illustrates the research scope and objective of the rule checking method.

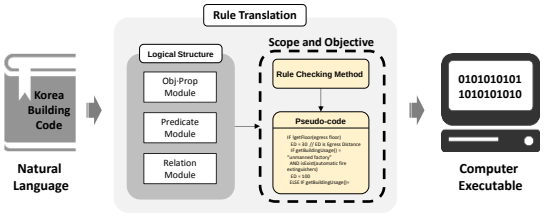

Figure 1. Overview logical structure-applied rule translation and the research scope of the rule checking method

Instead of employing complex implementation-centered programming language, combination of the three modules generates intermediate pseudo-code. The pseudo-code is later to be parsed into computer executable format such as XML or script language.

3 Rule-making and Representation Approaches

3.1 Current Rule-making Approach

Rule translation has been a challenging step for the process of the rule-based checking system [11]. Tracks of research activities in this area are as follows.

CORNET e-plan check, developed by Singapore BCA, is one of the earliest code checking efforts. It handles requirements on building control, barrier free access, fire code, environmental health, household, public housing and vehicle parking. CORNET rules were hard coded in computer programming language [9].

GSA project performed circulation rule checking on the US Court house design. To interpret the circulation requirements into computer executable rule, the research team predefined eleven parameters and processed each requirement sentence accordingly [12]. Because the parameters are specialized for circulation rules, it is impracticable to adjust to other types of rules.

Solibri Model Checker (SMC) is one of widely used model checker. It offers Rule Set Manager and allows users to define their own rule. By combining predefined rules checking libraries, users can make their own rule set [13].

Building code is always open to amendment. The hard coded way of making rule is fragile to modification and needs programming experts to manage rule. In other words, there is limitation in accessing rule-making. In addition, Building Code is not limited to certain type of rules but covers various issues. Therefore, universal way of making rule is necessary. Although SMC rule set manager enables various rule- making, its rule checking library is developer-centered. Without profound programming knowledge, it is hard to make rule set properly. Moreover, SMC predefines parameters for each rule. Parameters that are not predefined cannot be checked without being hard coded into Java language.

3.2 Law Sentence-based Approach

From the current rule-making methods, two challenges were found in rule-making. First, rule-making should not be limited in certain type of rules. Secondly, developer-centered methods are difficult to be used without programming knowledge. Building permit includes various types of rules and its automated compliance checking is accessible to novice programmers. Therefore, universal and intuitive way of rule-making is necessary. Rule checking methods are derived from law sentence-centered approach. It has intuitive naming convention and directly corresponds to the content of the law sentences. Therefore, novice programmers such as architects, designers, reviewers and anyone who wants to conduct compliance checking are able to write and read the pseudo-code.

Korea Building Code consists of article-clause-Ho-Mok structure. Single article delivers regulation of single issue. As the article break down into Mok level, which is the most segmented unit, single law sentence represents single requirement in general. We split 48 Building Code articles into 468 law sentences to derive high level rule checking methods.

3.3 High Level Method-based Representation of Law Sentences

The rule checking method is high level method. Unlike implementation level method for rule execution in model checker, it represents verbal phrase in law sentences for rule translation. The rule checking method has clear and intuitive name that matches with certain words in law sentences. For instance, if a law sentence asks to get the floor area ratio, a method named getFloorAreaRatio() is used to represent this sentence. Rule-making users can easily notice the usage of the method from its name. However, for the low level

implementation of this method in model checker, complicated calculation is hidden behind. Therefore, the high level method getFloorAreaRatio() is distinguished from low level methods used to calculate floor area ratio from a BIM model. Table 1 shows the list of high level rule checking methods derived from the scope of the Korean Building Codes.

Table 1. List of rule checking methods derived from the selected building code

Representative Method	Extended Method
getObject()	getSpace() getWindow()
isExist()	
getObjectCount()	getSpaceCount() getElementCount() getBuildingStroriesCount()
getProperty()	getDoorType() getElevLiveLoad()
getMaterial()	getMaterial type()
getObjectUsage()	getSpaceUsage() getBuildingUsage() getLandUsage()
getObjectHeight	getSapceHeight() getElementHeight() getBuildingHeight()
getObjectLength()	getSpaceWidth() getElementWidth()
getSpaceArea()	getSpaceAreaMax() getTotalFloorArea() getBuidlingToLandArea()
getElementArea()	getWindowArea() getDoorArea()
getObjectGradient()	getElementGradient()
getMaterialType()	
getSpaceIllunimance()	
getObjectStructure()	
isFireResistant	
isFireProof	
isfireCompartment	
hasObject()	hasSpace() hasElement()
getObjectDistance()	getSpaceDistance() getElementDistance()
isConnectedTo()	
isExternal()	
isAccessible()	
isAdjacent()	
isGoThrough()	
getDoorSwingDirection()	
isEgressDirection()	

Although the shape of the proposed methods associate with the verbal phrase, it need to be logically combined to fully deliver the meaning of the predicate in law sentences. The basic form of logical combination is as follows.

FUNC (PARAMS) OPERATOR VAL

1. FUNC() stands for high level rule checking methods.
2. PARAMS stands for parameters of the rule checking methods. Object and its property of building element, condition of checking or method itself can be parameters.
3. OPERATOR denotes comparison operators
4. VAL stands for value. Explicit value is substituted in left operand. According to the content of the check, the value varies from collection of objects, numeric, Boolean, string or method itself.

By Comparing left operand with right operand using comparison operators, logical combination composes a full meaning of predicate in the law sentence. Table 2 shows the example of logical combination applied to actual building code. The underlined phrases are reconstructed into logical combination.

Table 2. Example of representation of logical combination

Law sentence	[Article 64 Clause 1] A project owner of a building with six or more floors and a total floor area of 2,000 square meters or more shall have an elevator installed therein.
Logical Combination	getFloorCount() >= 6 getTotalFloorArea()>=2000m² isExist(elevator)=True

4 Rule Checking Method and its Classification

4.1 Object and Property-based Rule Checking Method

While the method represents verbal phrase, object and its properties represent building related noun phrase in law sentences. The rule checking method is inseparable from the objects and properties of building element, since they perform as parameters. The objects and properties are also law sentence-centered. They are not based on standard such as IFC, but on building code. For example, in the Korean Building Code, there are various terms regarding stairs such as escape stairs, special escape stairs, direct stairs, winding stairs etc. In the IFC schema, however, there is only IFCStair entity for stair objects. Therefore, an issue of mapping arises. The rule checking methods perform as a bridge to match objects and properties in requirement and IFC model.

Figure 2 illustrates the relationship between standard, IFC instance model, and building code. The role of rule checking methods is a connector between code requirement and instance model.

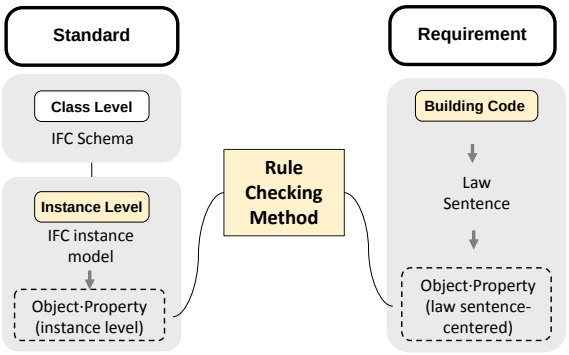

Figure 2. The role of rule checking method in relating IFC model instances and requirements

Law sentences contain objects and properties that are only recognizable in the instance level. For instance, rule checking on counting a certain object is only available once the building model is generated. Therefore, the rule checking method covers objects and properties not only handled in the class level (e.g. IFC schema) but also generated in the instance level (instantiated building model).

4.2 Method Classification

The method classification consists of three-level hierarchy. Based on the law sentence-centered object and property, each level has been classified. First, level 1 is classification of instance type. In this level, methods are divided into two groups: whether methods return value concerning property or not. For example, 'isExist()' method, which checks the existence of a certain object is included in the 'Object' group. The 'Object' group has nothing to do with the object's property. On the other hands, 'getMaterial()' method, which returns information about object's material belongs to the 'Object•Property' group. This group inevitably relates to object's property. Secondly, level 2 classifies the types of property. Because the 'Object' group does not concern with property, only the 'Object•Property' group is classified in this level. Thirdly, level 3 specifies the content of checking. In this stage, representative rule checking methods are defined. Table 3 illustrates the three-level hierarchy and detailed contents of the method classification.

Table 3. Hierarchical structure of the method classification

Level 1	Level 2	Level 3
Object		Query object
		Check existence
		Count object
Object • Property	Basic property	Get Property
	Derived property (geometry)	Get object's height
		Get object's width
		Get object's area
		Get object's gradient
	Derived property (complex)	Get space's illuminance
		Get object's structure
		Get finish material type
		Check firefighting related properties
	Relation	Check inclusion
		Check distance
		Check physical connection
		Get path
		Get direction

The 'Object' group is classified into querying object, checking existence and counting object in level 3. Figure 3 shows the overview of 'Object' group

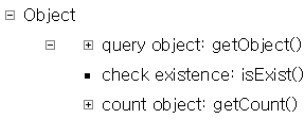

Figure 3. Classification of object axis

The 'Object•Property' group subdivides diversely rather than the 'Object' group. In the level 2, object's property is classified into three categories: basic, derived and relational property. The basic property is default property that object has when it is created by BIM authoring tools. Name, usage, material of an object is an example of basic property. Some of basic properties are automatically generated and the others need to be filled by users manually. As shown in the figure 4, getProperty() is the representative method for the basic property. It queries certain property of an object defined in the parameter.

Figure 4. Basic property and its representative method

As its name shows, the derived property is drawn

from calculation of property. The derived property is categorized into two groups. One is the property concerning geometric values such as height, width, area and gradient. The other is the property that should be inferred from other properties. This kind of property is named complex-derived property. The information needed to check complex-derived property does not directly exist in the building model. There are two ways of implement method for the property. One is to force designers to fill in the information in the BIM model using guidelines. Although this way is clear and explicit, it increases the complexity of modeling work. The other way is using programme and logic to derive information in implementation level. Figure 5 lists the representative methods within derived property.

- Derived property
  - Geometric
    - get an object height: getObjectHeight()
    - get an object width: getObjectLength()
    - get an object area: getObjectArea()
    - get gradient of a element: getElementGradient()
  - Complex
    - get space Illuninance: getSpaceIlluminance()
    - get object structure: getObjectStructure()
    - get object's Finish Material: getMaterialType()
    - check whether fire resistant: isFireResistant()
    - check whether fire proof: isFireProof()
    - check whether fire compartment : IsFireCompartment()

Figure 5. Derived property

In law sentences, there are properties that are not inquired with single object but with relation between multiple objects. This kind of property is relational property. Relations about inclusion, distance, physical connection, path and direction are examined. Each item is subdivided into specific rule of checking. For example, relation about path includes three methods. The methods are isAccessible(), isAdjacent() and isGoThrough(). As their names suggest, the methods check space accessibility, adjacency and whether path goes through specific space. The figure 6 shows the categories of relational property.

- Relation property
  - check relation of inclusion: hasObject()
  - get distance between objects: getObjectDistance()
  - check physical connection relation
  - get path
  - get direction

Figure 6. Relational property

The representative methods defined in the level 3 are subdivided into extended methods according to specific objects and properties. For example, representative method getObject() extends to getSpace(), getFloor(), getWall(), getWindow().

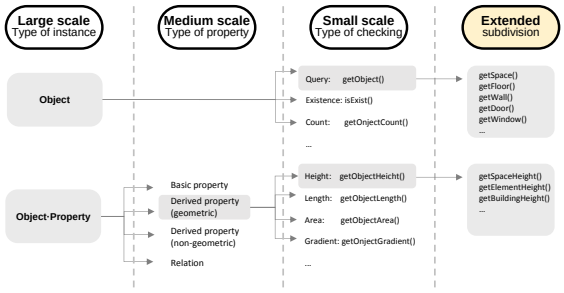

Figure 7. Extension of representative method

Naming convention for method extension is as follows. Every method should start with get, is or has. Methods start with 'get' queries exact object or value and return a collection of objects or numeric. Those start with 'is' and 'has' check condition and return Boolean.

The specific naming rule for each type is as follows.

1. get+[Object]+[Property]
   1) get+Object
      Ex.) getObject(), getSpace(), getDoor()
   2) get+Property
      Ex.) getProperty(),getMaterial()
   3) get+Property+Property
      Ex.) getPathDirection(), getMaterialType()
   4) get+Object+Property
      Ex.) getSpaceCount(), getFloorUsage()
   5) get+Object+Property+property
      Ex.) getWallMaterialType()

2. is+ [Object | property]+[Predicate]
   1) is+Predicate
      Ex.) isExist(), isSameDirection()
   2) is+Property
      Ex.) isFireResistant(), isFireProof, isAccesible()
   3) is+Object+Predicate
      Ex.) isWallExist(), getSpaceCount()

3. has+Object
   Ex.) hasObject(), hasSpace(), hasElement()

4.3 Extensibility

The rule checking methods we introduced in this paper are derived from a portion of entire building permit requirements stated in Korea Building Code. As a range of targeted code broaden, new objects and properties may appear. In consequence, new methods and extended version of existing methods will be needed. There are two

directions of methods extensibility: lateral extensibility and vertical extensibility. Lateral extensibility means extension of object type. On the other hand, vertical extensibility is an extension of property type. Since method classification consists mainly of classification of properties, advent of new property leads to creation of new rule checking methods.

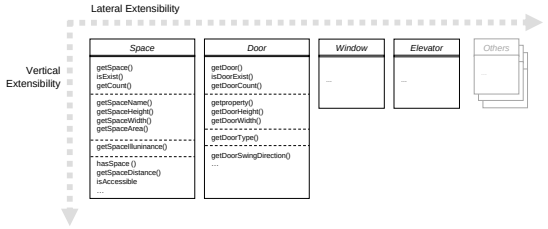

Figure 8. Two way extensibility of rule checking method

5 Demo: Sentence to Method

With the rule checking method, we represented some law sentences into pseudo-code. This code is an intermediate code between natural language and computer executable format. Although it is logically restructured code, it is easily understandable to human. The following table 4, 5, 6 show demonstration of sentences to methods.

Table 4. Demonstration 1

Law sentence	[Building Act Article 64 Clause 1] "A project owner of a building with six or more floors and a total floor area of 2,000 square meters or more shall have an elevator installed therein." [14]
Pseudo-code	IF **getBuildingFloor()** >=6 AND **getFloorArea()** >=2000 ㎡ **isExist(elevator)** END IF

Table 5. Demonstration 2

Law sentence	[Enforcement of the Building Act Article 34 Clause 1] "On each floor of a building, direct stairs leading to the shelter floor or the ground other than the shelter floor shall be installed in the way that the walking distance from each part of the living room to the stairs is not more than 30 meters: Provided, That in cases of a building of which main structural part is made of a fireproof structure or non-combustible materials, the walking distance of not more than 50 meters may be established, and in cases of a factory prescribed by Ordinance of the Ministry of Land, Infrastructure and Transport, which is equipped with automatic fire extinguishers, such as sprinklers, in an automated production facility, the walking distance of not more than 75 meters may be established." [14]
Pseudo-code	IF !**getFloor(egress floor)** ED = 30 // ED is Egress Distance IF **getBuildingUsage()** = "unmanned factory" AND **isExist(automatic fire extinguishers)** ED = 100 ELSE IF **getBuildingUsage()** = "factory" AND **isExist(automatic fire extinguishers)** ED = 75 ELSE IF **getMaterialType(main structural part)** = "non-combustible materials" AND **isFireResistant(main structural part)** ED = 50 END IF **getSpaceDistance(living room, stair, MRP)** <= ED END IF

Table 6. Demonstration 3

Law sentence	[Enforcement of the Building Act Article 90 Clause 1] "Emergency elevators (including the platform and shaft of an emergency elevator; hereafter the same shall apply in this Article) shall, under Article 64 (2) of the Act, be installed in buildings of which height exceeds 31 meters in not less than the number according to the criteria in each of the following subparagraphs: Provided, That the same shall not apply to cases an elevator installed under Article 64 (1) of the Act is of the structure of an emergency elevator: 1. Buildings of which height exceeds 31 meters and of which largest floor area among the floor areas of each floor is not more than 1,500 square meters: Not less than one unit; 2. Buildings of which height exceeds 31 meters and of which largest floor area among the floor areas of each floor exceeds 1,500 square meters: One unit plus one unit for every not more than 3,000 square meters in excess of 1,500 square meters." [14]
Pseudo-code	IF **getBuildingHeight()** > 31 AND **getNumberOfObject(emergency elevator)** **getFloor(height over 31m)** = A

```
        IF getSpaceAreaMax(A) <= 1500
          getNumberofElement(elevator) >= 1
        ELSE
          getNumberofElement(elevator)      >=
          (1+getSpaceAreaMax(A)/1500)

        END IF

    END IF
```

The pseudo-code is generated through the KBimLogic software. KBimLogic performs translation of building code into computer executable form. It offers user interface for users to manually restructure building code. The intermediate code generated by users is then parsed into computer executable code such as XML or script language. KBimLogic is under development as a part of government funding project. It will be used together with other software to develop an automated building permit system for Korea government.

6 Summary

In this paper, we introduced the rule checking method and its application on Korea Building Code related to building permit requirements. The rule checking method is developed with law sentence-centered approach. It is a high level method that is directly mapped with verbal phrase in the law sentence. The classification of the rule checking method is based on the object and property of a building. There is three-level hierarchy in method classification. Level 1 divides type of instance, Level 2 classifies type of property, and Level 3 specifies the content of checking. From the level 3 representative method is defined. The representative method is subdivided into various methods according to specific objects and properties to check.

The rule checking method introduced in this paper is developed from a part of Korea Building Code. As the application of logical structure extends to the rest of building permit requirements, the method will extend in two directions: 1) lateral extensibility, which means the extension of object type, and the 2) vertical extensibility, the extension of property type. We represented actual law sentences with combination of high level rule checking methods. The pseudo-code is an intermediate code and later to be parsed into computer executable format.

The rule checking method is developed as a part of logical structure for KBimLogic. KBimLogic is a software that translates Korean building permit requirements into computer executable format. Together with other softwares, the KBimLogic will establish automated building permit system for Korea government.

Acknowledgement

This research was supported by a grant (14AUDP-C067809-02) from Architecture & Urban Development Research Program funded by Ministry of Land, Infrastructure and Transport of Korean government.

References

[1] Jin-Kook Lee, Jaemin Lee, Yeon-suk Jeong, Hugo Sheward, Paola Sanguinetti, Sherif Abdelmohsen, Charles M. Eastman, Development of space database for automated building design review system, *Automation in Construction*, 24:203-212, 2012.

[2] C.M. Eastman, P. Teicholz, R. Sacks, K. Liston, *BIM Handbook—A guide to Building Information Modeling for Owners, Managers, Designers, Engineers, and Contractors*, John Wiley & Sons Inc., Hoboken, NJ, United States of America, 2008.

[3] C. Han, C. J. Kunz, and K. H. Law. Making Automated Building Code Checking a Reality, *Facility Management Journal*, 22-28, 1997.

[4] Greenwood, David, Lockley, Steve, Malsane, Sagar and Matthews, Jane Matthews, Automated compliance checking using building information models, In *Proceeding of The Construction, Building and Real Estate Research Conference of the Royal Institution of Chartered Surveyors*, Paris, France, 2010.

[5] C. Eastman, Automated assessment of early concept design, article In *Architectural Design Special Issue: Closing the Gap*, 79(2):52–57, 2009.

[6] Johannes Dimyadi, Robert Amor, Automated Building code Compliance Checking Where is it at?, In *Proceedings of CIB WBC* 2013, pages 172-185, Brisbane, Australia, 2013.

[7] GSA, GSA courthouse program. On-line: http://www.corenet.gov.sg/integrated_submission/esub/esub_faqs.html, Accessed: 29/01/2015.

[8] Lan Ding, Robin Drogemuller, Julie Jupp, Mike A Rosenman, Jhon S Gero, Automated code checking, In *Proceeding of Clients Driving Innovation International Conference*, pages 25–27, Surfers Paradise, Qld, Australia, 2004.

[9] C. Eastman, Jae-min Lee, Yeon-suk Jeong, Jin-kook Lee, Automatic Rule-based Checking of Building Designs, *Automation in Construction*, 18(8):1011-1033, 2009.

[10] L. Ding, R. Drogemuller, M. Rosenman, D.

Marchant, J. Gero, Automating code checking for building designs, In *Proceeding of Clients Driving Construction Innovation: Moving Ideas into Practice*, pages 113-126, Brisbane, Australia, 2006.

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

[12] Jae-min Lee, *Automated checking of building requirements on circulation over a range of design phase*, ph.D. Dissertation, Georgia Institute of Technology, 2008.

[13] Solibri. Solibri Model Checker. On-line: http://www.solibri.com/products/solibri-model-checker/, Accessed: 29/01/2015.

[14] Korea Legislation Research Institute, Building Act and Enforcement of the Building Act, On-line: http://elaw.klri.re.kr/kor_service/main.do, Accessed: 29/01/2015.