CONVERTING KBIMCODE INTO AN EXECUTABLE CODE FOR THE AUTOMATED DESIGN RULE CHECKING SYSTEM

JAEYEOL SONG[1], JINSUNG KIM[2] and JIN-KOOK LEE[3]
[1,2,3]*Department of Interior Architecture & Built Environment, Yonsei University, Republic of Korea*
[1,2]*{songjy92|jinsungkim212}@gmail.com* [3]*leejinkook@yonsei.ac.kr*

Abstract. This research aims to describe an implementation approach for a translator of KBimCode as a part of a BIM-enabled automated design rule checking system. KBimCode is an explicit and computer-readable form written in a scripting language to represent Korea Building Act sentences. KBimCode separates the rule-making process that conventionally dependent on rule-checking software. Based on the approach, KBimCode implemented with its own logic rule components and has been managed with the database. On the other hand, there are several rule-checking software executed with their own rule set. Translating natural language rules into the rule set of each software and translating a rule of specific software into others require a lot of manual effort. The manual translation also hinders interoperability between rule checking software. We address the problem with developing the KBimCode translator for each rule checking software. In this research, we focused on translating KBimCode into an executable code of specific rule checking software, named KBimAssess. KBimCode translator will integrate the rule-making and rule-checking process, which means various stakeholders, even who are not familiar with programming, can easily conduct BIM-enabled rule checking by utilizing KBimCode. Furthermore, the implementation of KBimCode translator is expected to contribute to the enhancement of interoperability between various rule-checking applications.

Keywords. Automated design rule checking; Building information modeling (BIM); Executable code; Language translator; KBimCode.

1. Introduction

Research and development in automated rule checking of building design have been conducted continuously and gained more interests as BIM (building information modeling) provides computable information for building objects (Solihin and Eastman 2015). In order to check code compliance with information from a given BIM model, building codes or design requirements have to be represented with explicit and computer-readable form. Building codes and design requirements are written by professionals and applied to domain-specific tasks with experts' interpretation. Therefore, the computerization of natural

Intelligent & Informed, Proceedings of the 24th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2019, Volume 1, 795-804. © 2019 and published by the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), Hong Kong.

language-based rules or requirements has been regarded as a challenging process (Nawari 2012). There have been many efforts to translate implicit natural language rules into explicit and computer-readable form (Solihin et al. 2017, Eastman et al. 2009). KBimCode is a domain specific language for representing Korean Building codes. KBimCode can alleviate the labor and cost of the rule-making process by translating natural language-based building regulations with a logic rule named KBimLogic (Lee et al. 2015).

Since the main purpose of KBimCode is to represent Korean Building Act in an intuitive way, KBimCode is based on the software independent approach to representing natural language rules as computable rules. Various rule checking applications have been developed apart from the development of KBimCode. Each rule checking application uses its own data structure and checking functions. In principle, KBimCode can interoperate with any rule checking software thanks to the neutral and software independent features. The implementation of rule checking application and KBimCode, however, has been proceeded separately. Interoperation of the rule checking application and KBimCode requires the development of KBimCode translation module that maps the KBimCode with the low-level execution code of rule checking software. In this research, we focused on an approach to translating KBimCode into the target executable language.

In KBimCode syntax, rules defined in building act sentences are represented with three types of statements: 1) KBimCode Object Model (KOM), 2) Conditional Statement, and 3) Arithmetic Logic Units (ALUs). Since KBimCode is computer-readable language and it has a formal definition, we can easily implement KBimCode parser that can classify the type of each line. Each statement has to be translated with its semantic meaning. In this research, we handle how to appropriately translate each statement by reflecting on the meaning of KBimCode statements and propose some frameworks for generalization. The proposed frameworks enable KBimCode to interoperate with diverse rule checking software, as mentioned above. We briefly demonstrate an example of implementation for translating KBimCode in the Python-based executable code which used in a rule-checking software named KBimAssess.

2. Implementation approach to converting KBimCode

2.1. OVERVIEW OF KBIMCODE TRANSLATION

The purpose of the KBimCode translator is to enhance the interoperability of regulatory data expressed in KBimCode. Various research and development projects have implemented rule checking tool such as Solibri Model Checker (SMC), EDM ModelServer, FORNAX, KBimAssess, etc. All of them make their own rule set and tried to provide user-friendly interfaces to define and manage the rules (Solihin et al. 2017). The problem of conventional rule checking implementation is on the dependency to rule checking software. By using their own rule set, end user have to translate building regulations into computer-readable form by software. KBimCode was developed to solve this problem, separating the rule-making process from the specific rule checking software. The main purpose of KBimCode is to represent the natural language-based building regulations into an explicit form. However, the role of the KBimCode has to extend to the rule checking process, for practical utilization in automated rule checking process. KBimCode translator plays an important role in extending the utilization of KBimCode. Development of KBimCode translator for respective rule checking software enables to execute rule checking with KBimCode, and it also can contribute to interoperate different rule checking software or even BIM authoring tool.

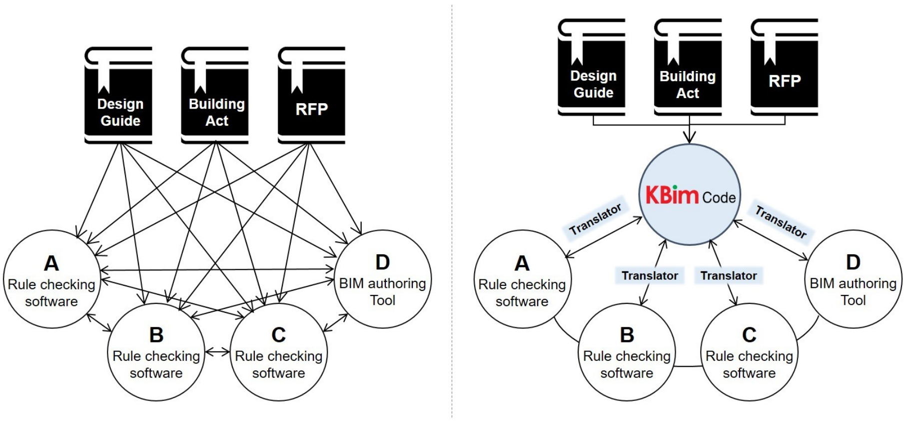


Figure 1. Overview of KBimCode Translator development -Left: conventional rule-making and rule set translation process, Right: Interoperated rule checking process with KBimCode and KBimCode translator.

The KBimCode translation is based on an approach with model-driven language translation engine (Parr 2009). Likewise other language implementation mechanism, KBimCode translation system is basically composed with three modules 1) syntax parser, 2) semantic analyzer, and 3) language generator. The role of the syntax parser and semantic analyzer is to understand the building regulatory information expressed in input KBimCode. Semantic analyzer returns an intermediate representation which transfer the regulatory information expressed in KBimCode and language generator consume the intermediate representation for generating target languages. The intermediate representation is a neutral format for representing a regulatory information, which can be expressed in various language form by the language generator. In other words, KBimCode translator can translate KBimCode into various language just changing the generator module.

2.2. LEXICAL ANALYSIS AND SYNTAX PARSING

Language recognition is a critical step for any language translation (Parr 2009). Each input phrase or word has to be recognized by their grammatical usage. For instance, the computer should be able to recognize what the word “wall” is intended for. It can be used for a variable name for variable declaration or object classes which stands for wall class. Lexical analysis and syntax parser analyze the inputted script code according to pre-defined grammar and identify the elements. Describing detail mechanism of a lexical analyzer and syntax parser is not the scope of this paper, instead, we describe how KBimCode statements are classified by their grammatical usage in Chapter 3. Each KBimCode statements are converted into a defined data structure suitable for representing building object data.

2.3. SEMANTIC ANALYSIS

The main purpose of a semantic analysis is identifying the semantic relation of each element. Syntax parser is mainly focusing on identifying the input phrases and generates parsing data. Using the parsing data, the computer can understand the grammatical structure of phrases and elements. However, they have no idea about relations between each element. For example, the computer know “wall” stands for building object and “height” is the word for properties of the object respectively, but it can’t recognize that the “height” is used for an associated property of the “wall” object. Furthermore, building rules include a lot of conditional relations between the rules or even in a single rule. In order to translate input KBimCode correctly, and without losing the semantic purpose of rule checking, the semantics of each element has to be clarified.

KBimCode Object Model statements are used to clarify the user-defined building objects which have specific properties or constraints. Querying those specific building objects and grouping them needs appropriate algorithms and frameworks. When translating Conditional Statement such as IF-THEN-ELSE statements, the checkpoint that judges whether the condition is satisfied or not, should be specified. ALUs is an atomic unit of rule checking, comparing values from given BIM model and required values defined in regulations. High-level methods, a component of ALUs, are needed to map with low-level methods to query the property values from given BIM model. Therefore, classifying the intent of a high-level method is critical in translating ALUs. In other words, the computer has to understand which properties have to be checked.

2.4. TARGET LANGUAGE GENERATOR

The target language generator returns a translated execution code by facilitating the intermediate representation. There should be a mapping table for generating corresponding building objects and properties data that could be executed in a target rule checking software. Based on the mapping table we can translate the KBimCode using rules that match the regulatory data to that of the target software. There might be an additional loop or conditional statement for querying the building objects or their associated properties due to the abstracted expression of the high-level method in KBimCode. Developers also have to build the logic of low-level functions to translate high-level methods which need an additional loop or condition statements. In this paper, we don’t generalize the mechanism of language generator since the implementation methods are varies by target software and languages. Instead, we demonstrate an example of implementations of the mapping table and translation test in chapter 4, with the specific target languages.

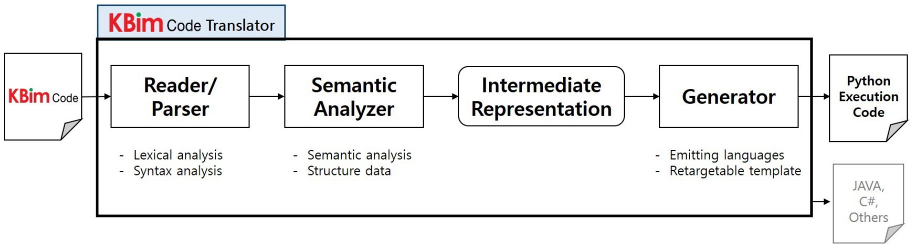


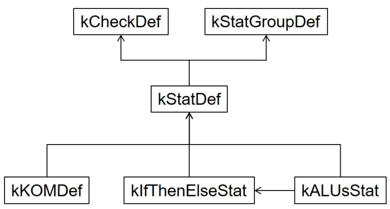
Figure 2. KBimCode translation process with KBimCode translator.

3. Implementation of KBimCode translator

3.1. KBIMCODE SYNTAX PARSER

The language design of KBimCode is based on a general-purpose language, JAVA, and its formal definition is described with EBNF-based ANTLR rule (Park et al. 2016). ANTLR is a language parser generator which used for translating or implementation of languages. We can easily implement language parser with ANTLR. As mentioned in chapter 2, we focused on how to translate parsing results for use in subsequent steps.

Figure 3 shows a hierarchy of KBimCode statements. A check declaration, a statements group, and a KBimCode statement are key components of KBimCode language. The check declaration is a fundamental component of KBimCode, which declare and contain the statements of rules. The statements group is used for grouping several statements. By grouping the statements, the statements group make KBimCode more intuitive when representing the conditional relations (If/Then/Else). Both components are composed of a set of KBimCode statements. KBimCode is categorized with three types of statements: KBimCode Object Model (KOM), Condition statement, and Arithmetic Logic Units (ALUs).




```
kCheckDef
     :    CHECK '(' kCheckParamDef ')'
          '{' LT* kStatDef LT* '}'
     ;
-----------------------------------------------
kStatGroupDef
     :    kStatGroupVar LT*
          '{' LT* kStatDef LT* '}'
     ;
-----------------------------------------------
kStatDef
     :    kStatLines
     ;
kStatLines
     :    kStatLine+
     ;
kStatLine
     :    (kKOMDef | kIfThenElseStat | kALUsStat) LT*
     ;
-----------------------------------------------
LT // New lines
```

Figure 3. A snippet of a formal definition of KBimCode statements (Right) and relation diagram between grammar components of KBimCode (Left). .

KOM is an abstracted representation of building objects specified in Korean Building Acts. KOM is a borrowed concept from the BERA Object Model (BOM) (Lee 2011, Lee et al. 2015). KOM enable users to define the specific object or object groups which have specific constraints for rule checking. ALUs is an atomic unit for rule checking. It returns Boolean values, true or false, by comparing operands with the operator. Conditional statements represent IF-THEN-ELSEIF-ELSE logic. Statements described above also have their own formal definition respectively

3.2. SEMANTIC ANALYSIS

The semantic analyzer is in charge of analyzing the relation between the elements of parsing results and converting the information into the intermediate form. In Korean Building Act, targets of rule checking vary with the existence of objects, specific value of the property, and the relation between objects. There are complex relationships between building objects and its associated properties. Building objects and its associated properties are expressed with dot-notation in KBimCode for representing their relation intuitively. The examples of dot-notation are as follow.

```
1)Wall
2)Wall.isLoadBearingWall
3)Floor.Room
4)Floor.number
5)Wall.Structure.materialType
6)Wall.InteriorFinsh.Material.nonCombusitibility
```

Example 1)˜5) show variations of dot-notation. Example 1) stands for a single wall object and example 2) show the expression of an objects-property relationship, in this case, representing wall objects which is load bearing for the building structure. Example 3) shows an object - object relationship which has an inheritance relation. The user can simply define or query the objects which are dependent on a specific object thanks to this dot-notation. Example 4) is the expression of object-quantity relation, which is used for querying the object with a specific quantity. The dot-notation can be extended with relations mentioned above, as shown in example 5) and 6). We classify the variance of dot-notation expression and make a rule for converting relations of the object and its properties in an intermediate representation which used for generating a target language.

4. Preliminary Experiments of Converting KBimCode into target executable code

4.1. MAPPING KBIMCODE COMPONENTS TO TARGET SOFTWARE LANGUAGES

In this research, we implemented a KBimCode translator module for KBimAssess which is one of the rule checking software developed for automated building admission system in Korea. KBimAssess uses IFC file model for rule checking. It means that the KBimAssess data structure for building objects is based on the IFC data structure. Therefore mapping the building object data of KBimCode with KBimAssess is almost same as the mapping for IFC data scheme. The checking method is the most important components in the rule checking process, therefore mapping of method between KBimCode and KBimAssess is also critical for translation. KBimAssess have not only the method function for querying simple objects or its properties, but also have the method for calculating complex geometry or reasoning the relations of objects. We analyzed the implemented method of KBimAssess and mapping them with checking method of KBimCode.

KBimAssess is basically developed with C sharp languages, but it also uses python-based intermediate code which end user can edit for their purpose. The target language of KBimCode translator for KBimAssess, therefore, is a python-based intermediate code of KBimAssess. We made a mapping table for building objects, properties and checking methods. By structuring the mapping table, we can identify which elements are not defined in IFC data, or which method is required for implementing for rule checking of Korea Building Acts (see Figure 4).

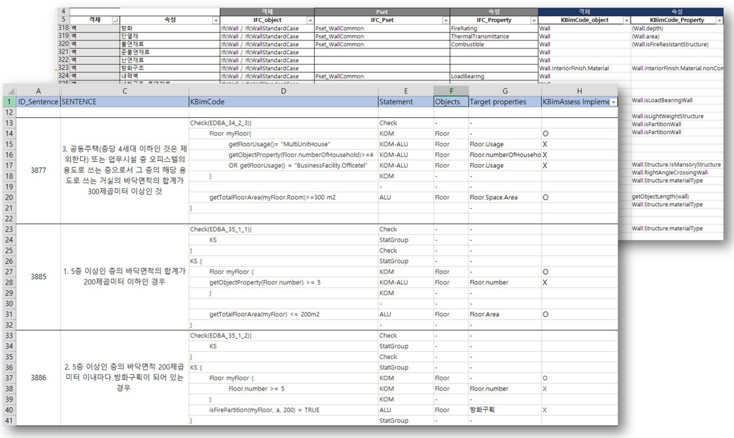


Figure 4. KBimCode-KBimAssess mapping Table. 1) Above: mapping table for objects and properties data, 2) Below: mapping table for KBimCode statements and methods .

4.2. TEST WITH MANUAL TRANSLATION AND PROTOTYPE IMPLEMENTATION

As developing the KBimCode translator module, manual translation tests have been carried out for validation of data mapping between KBimCode and KBimAssess. Figure 4 shows an example of the manual translation process and its results. The input KBimCode is composed of two statements groups, condition statements, and key statements and their IF-Then relation. The key statement defines the user-defined object, for querying floor which has floor area more than 500 square meters. And condition statement is an ALUs which check the width of the corridor placed in the defined floor. In KBimCode user-defined floor object are simply defined with one single line, but execution code uses for loop to find the corresponding floor object in a given building model. The execution code of KBimAssess also uses dot-notation for representing building objects and their associated properties, similar to KBimCode. “SELECT” method is a key method for querying building objects and properties. SELECT method querying the entire list of object or property values and software can go through each object or values in the list, using the loop statement. In this way, we can find a floor slab which has floor area more than 500 square meters with if statement after the loop. It reasons whether it goes to the next step or not. If the floor area of a slab is more than 500 square meter, then it goes to next step for checking the width of the corridor. Querying and reasoning about the width of corridors have almost the same process with querying the area of the slab. The results of translation are executed without a problem and return the rule checking results.

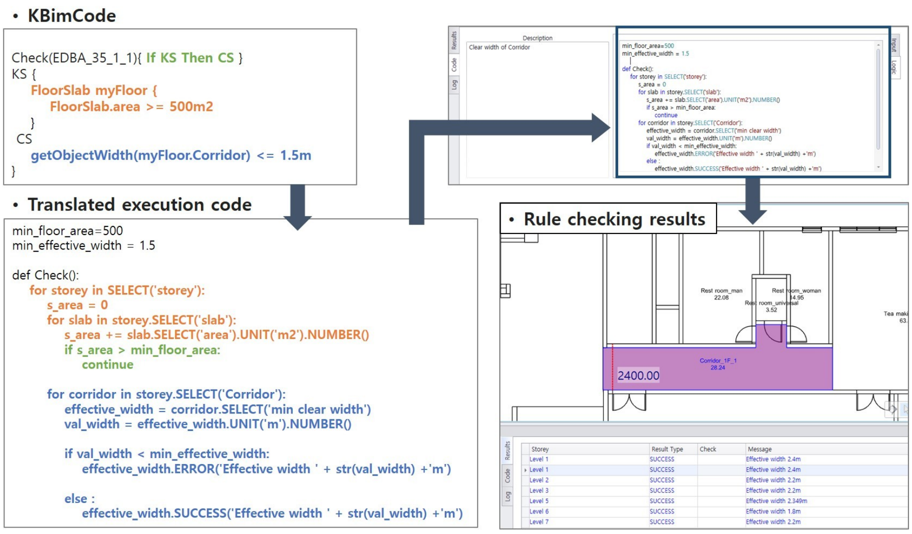


Figure 5. Manual translation process and rule checking results .

KBimCode translator module will be implemented as a part of KBimAssess plugin for authoring KBimCode. Figure 5 shows a prototype of KBimCode authoring plug-in in KBimAssess software. KBimCode authoring plug-in is connected with KBimCode DB, which enables the end user to search the Korean Building Act sentences in KBimCode format (1 in Figure 5). Using the KBimCode translator, users can translate KBimCode into KBimAssess execution code easily and directly use it for checking their own building model (2 in Figure 5).

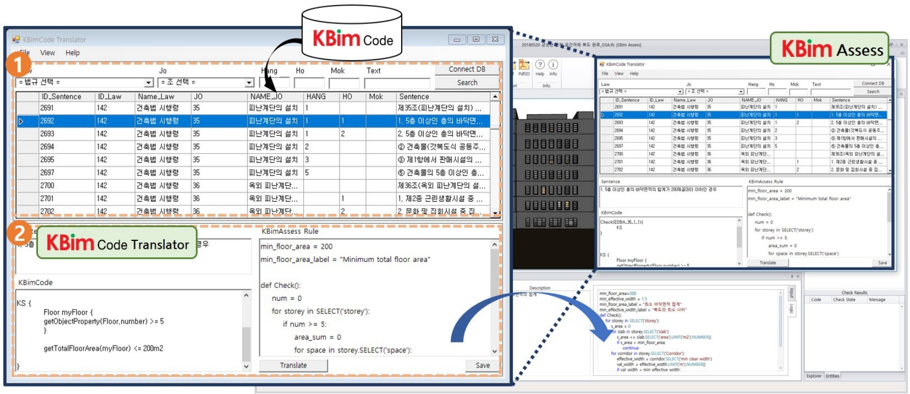


Figure 6. Prototype of KBimCode Authoring plug-in which contains KBimCode Translator module.

5. Conclusion

This research addresses the problems to be handled in converting KBimCode into an executable code of rule checking software. In implementing KBimCode translator, semantic analysis of KBimCode is critical for an accurate translation. We focus on how to interpret the semantic relation between building objects and their associated properties and briefly demonstrate the results of development.

Developing a translator of KBimCode integrates the rule-making and rule checking process. This will enhance the accessibility to automated design rule checking system for domain experts who are not familiar with programming. The software-independent approach of KBimCode and its translator maximizes the interoperability with various software. Developing the translator for various target languages, KBimCode can be used not only the rule checking software, but also the BIM authoring software such as Autodesk Revit, ArchiCAD. There is also the visual language version of KBimCode for translating building regulation more intuitively (Kim et al. in press). We also have a plan to develop the translator of the visual language of KBimCode with its script language, which contributes to lower the entry barrier of rule checking process to various stakeholders. Furthermore, the translation module can be implemented by utilizing deep learning and language processing technique. The utilization of intelligence technology translator module can translate not depended on translation rule, but based on its own translation mechanism with an amount of training data. This can alleviate labors and times consumed for developing the language mapping table and translation rules.

Acknowledgements

This research was supported by a grant(19AUDP-B127891-03) from Architecture & Urban Development Research Program funded by Ministry of Land, Infrastructure and Transport of Korean government.

References

Eastman, C.M., Lee, J.M., Jeong, Y.S. and Lee, J.K.: 2009, Automatic rule-based checking of building designs. Automation in construction, *Automation in construction*, **18**(8), 1011-1033.

Kim, H.Y., Lee, J.K., Shin, J.Y. and Choi, J.S.: 2018, Visual language approach to representing KBimCode-based Korea building code sentences for automated rule checking, Journal of Computational Design and Engineering, *Journal of Computational Design and Engineering*, **inpress**, 6.

Lee, J.K.: 2011, *Building Environment Rule and Analysis (BERA) Language*, Ph.D. Thesis, Georgia Institute of Technology.

Lee, J.K., Eastman, C.M. and Lee, Y.C.: 2015, Implementation of a BIM Domain-specific Language for the Building Environment Rule and Analysis, *Journal of Intelligent & Robotic Systems*, **79**, 507-522.

Lee, H.S., Lee, J.K., Park, S.K. and Kim, I.H.: 2016, Translating building legislation into a computer-executable format for evaluating building permit requirements, *Automation in construction*, **71**, 49-61.

Nawari, N.: 2012, The Challenge of Computerizing Building Codes in BIM Environment, *Computing in Civil Engineering*, **2012**, 285-292.

Park, S.K., Lee, Y.C. and Lee, J.K.: 2016, Definition of A Domain-specific Language for Korean Building Act Sentences as an Explicit Computable Form., *Journal of Information Technology in Construction*, **21**, 422-433.

Parr, T.: 2009, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*, Pragmatic Bookshelf.

Preidel, C., Daum, S. and Borrmann, A.: 2017, Data retrieval from building information models based on visual programming, *Visualization in Engineering*, **5**(1), 18.

Solihin, W., Dimyadi, J., Lee, Y.C., Eastman, C.M. and Amor, R.: 2017, The Critical Role of the Accessible Data for BIM Based Automated Rule Checking System, *Proceedings of the Joint Conference on Computing in Construction (JC3)*, Heraklion, Greece, 53-60.

Solihin, W. and Eastman, C.M.: 2015, Classification of rules for automated BIM rule checking development, *Automation in construction*, **53**, 69-82.