S. Chien, S. Choo, M. A. Schnabel, W. Nakapan, M. J. Kim, S. Roudavski (eds.), *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA 2016,* 73–82.

DEFINITION OF A DOMAIN-SPECIFIC LANGUAGE TO REPRESENT KOREA BUILDING ACT SENTENCES AS AN EXPLICIT COMPUTABLE FORM

SEOKYUNG PARK and JIN-KOOK LEE
Hanyang University, Seoul, Korea
seokyung.park529@gmail.com, designit@hanyang.ac.kr

Abstract. This paper aims to define the syntax of KBimCode Language as a domain-specific computer language to represent Korea Building Act sentences. KBimCode Language represents building permit requirements in Korea Building Act as explicit computable rules. KBimCode aims to accomplish the neutral and standardized way of rule-making in an easy-to-use syntax. This paper introduces the approach of language design and definition. The main concerns handled in the paper are: 1) features of building permit-related regulations in Korea Building Act are reflected in the strategy for the lexical and syntactic design of KBimCode Language; 2) specification of KBimCode based on the context-free EBNF notation is introduced; and evaluation of the language definition is performed. KBimCode is an ongoing project. Together with newly developed rule checking applications, KBimCode will establish automated design quality assessment system in Korea.

Keywords. Automated building permit system; automated design assessment; rule checking; rule-making; domain-specific language.

1. Introduction

Building Information Modelling (BIM) is increasingly used in the various fields of architecture, engineering, construction and facility management (AEC-FM) industry. As one of promising direction of BIM application, automated assessment of building design become available (Lee et al, 2012; Eastman, 2009). Conventionally, design assessment was done by manual and it was tedious, iterative, time-consuming, and error-prone process (Ding et al, 2006). BIM-enabled design assessment is expected to streamline this pro-

cess and increase efficiency and precision (Han et al, 1997; Eastman et al, 2009).

The design requirements are written in human language (natural language) and consequently embraces ambiguity and vagueness (Nawari, 2012). Therefore, the natural language rules need to be formalized into a set of explicit and computational rules (Malsane et al, 2015) to perform automated code compliance checking. It is a process of transforming design knowledge into a computable format and maximizing computer-executable semantics. As one of the rule-making solutions, this paper introduces domain-specific language called KBimCode. KBimCode is developed to translate building permit regulations in Korea Building Act into computable rules with ease of use and high fidelity. Among various issues regarding KBimCode, this paper mainly focuses on language design and definition. The following sections introduce 1) overview of KBimCode, 2) strategy for language design, 3) specification of KBimCode based on the context-free EBNF notation, and 4) language evaluation.

2. What is KBimCode?

The distinguishing feature of KBimCode is that it is a software-independent and standardized approach to the rule-making process. Some previous projects embedded design knowledge in rule checking tools. In such case, the rule-making process was integrated with the development of rule checking software. The resulting rules are inevitably dependent on specific proprietary software and closed BIM environment. On the other hands, KBimCode is an intermediate script language that is independent of rule checking tools. As Figure 1 illustrates, KBimCode is processed into open text files such as JSON (ECMA, 2015), XML (W3C, 2015), etc. Therefore, it can be used in various BIM assessment tools. The theoretical approach for generating KBimCode is described in (Lee, 2015).

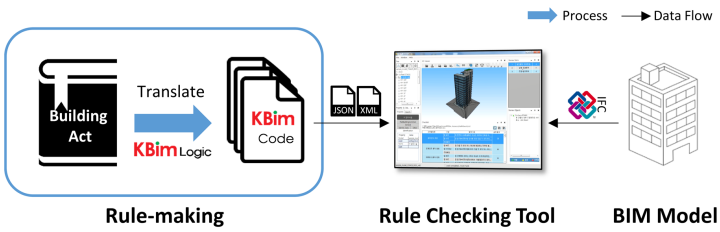

Figure 1. The scope of KBimCode within the entire rule checking process. Building Act sentences are translated into KBimCode and then processed into open text files to be imported to rule checking tools for evaluating building permit requirements.

Based on such background, KBimCode can be summarized as follows:

1) KBimCode can be classified as a domain-specific language, a programming language, and a rule language.
2) Its main functionality is to convert natural language rules into computable rules for automated code compliance checking.
3) Its target field is basically AEC industry, especially focusing on building permit issues in Korea.
4) KBimCode Language attempts to deal with building act in an intuitive way in order to define user-friendly computable rules.

The main target of KBimCode is Korea Building Act, especially building permit regulations. The entire regulations were analysed to capture general syntactic and lexical elements of language design. For example, Korea Building Act-specific building objects and their properties were clarified to be used as a lexicon of the KBimCode. Various expressions such as high-level methods or dot notation approach were devised in advance to describe specific conditions in the building act.

3. KBimCode language design

3.1. LEXICAL DESIGN

This section describes a lexical design strategy for basic tokens and idioms used in KBimCode Language. The fundamental strategy can be as follows:

1) A Building act can be split into atomic sentences (ASs), a type of declarative sentence that is either true or false. The AS is processed into translated atomic sentence (TAS) that can be expressed in a single S (subject) + V (verb) + O (object) structure.
2) KBimCode assumes that S is Korea Building Act-specific building objects and properties. They are specified into a dictionary and will be used as the fundamental lexicon.
3) To represent properties of KBimCode objects: dot-notation.
4) Predefined object names will be starting with upper case, while properties and user-defined names will start with lower case.
5) V + O structure can be pre-defined as basic methods, for example, getObect(), isExist(), getObjectDistance(), etc. The details of the pre-defined methods are described in (Park et al, 2015).

3.2. SYNTACTIC DESIGN

This section briefly previews a syntactic design style for KBimCode Language. A basic syntactic form of KBimCode can be represented as follows:

```
check(arg) {
```

```
    Statement;
    …
}
```

– Where `check` is a pre-defined method that declares the checking of a certain building act sentence.
– Where `arg` and `Statement` are non-terminal tokens.

The non-terminal token `arg` is pointing a building act sentence to be checked. The `Statement` is a rule defined in the building act sentences. The statement types can be categorized as follows:

- Arithmetic Logic Units (ALUs)
- Conditional Statement
- KBimCode Object Model (KOM)

Originally, ALU is a digital electronic circuit that performs arithmetic and bitwise logical operations (Wikipedia, 2015). In this paper, ALU stands for a declarative clause that performs the atomic unit for rule checking. Structure of the ALU is detailed in the next section. As for Korea Building Act, content of a single sentence consists of two parts: a checking condition and a checking content. They are joined using IF-THEN-ELSEIF-ELSE logic to form conditional statement. KBimCode Object Model (KOM) is one of the key features of KBimCode Language. It is a human-cantered abstraction of building objects specified in the building act. By using KOM, user can define virtual objects with desired rules as constraints. The defined KOM can be applied to the various locations within KBimCode.

A set of statements can be grouped and used as an object. The syntactic form of the statements group is as follows:

```
var {
    Statement;
    …
}
```

– Where `var` and `Statement` are non-terminal tokens.

The statements group is declared outside of the basic syntactic form and its name is used as a variable in the form. By using the statements group, KBimCode can be written in a clear and simplified way.

4. KBimCode language definition

4.1. OVERVIEW

This chapter defines the syntax of KBimCode language grammar. The formal definition described in this chapter is EBNF-based ANTLR rule (Parr,

2008). ANTLR is a language parser generator. Non-terminals starting with lower case are syntactic rules and non-terminals starting with upper-case are lexical definition. As mentioned in the previous section, KBimCode Language basically has three key components as follows:

- Check declaration
- Statements Group declaration
- KBimCode statement

The Check declaration and the Statement Group declaration consist the syntactic form of KBimCode, while KBimCode Statement is subordinate to the other two components. These three components are defined as: [1] kCheckDef, [2] kStatGruopDef, and [3] kStatDef. Most of all, a non-terminal syntactic rule kKBimCodeProgram is the starting point of KBimCode Language definition. A lower case 'k' means KBimCode and the rest of alphabets are simplified token to represent each syntactic components.

```
kKBimCodeProgram
    :       kCheckDef? kStatGroupDef?
    ;
```

[1] Definition: kCheckDef

```
kCheckDef
    :       CHECK '(' kCheckParamDef ')'
            '{' kStatDef '}'
    ;
```

[2]Definition: kStatGroupDef

```
kStatGroupDef
    :       kStatGroup '{' kStatDef '}'
    ;
```

In these syntactic rules, the lexical rule `CHECK` stands for the pre-defined method "check" that instantiates checking of a target building act sentence. The `kCheckParamDef` defines identifier tokens for the target sentence. The `kStatGroup` defines a user-defined variable name of the Statement Group.

4.2. KBIMCODE STATEMENT

The KBimCode statement has three types of statements and each type is defined as [3-1] kALUsStat, [3-2] kIfThenElseStat, and [3-3] kKOMDef. The definition of KBimCode statement is as follows:

[3] Definition: kStatDef

```
kStatDef
```

```
    :      kStatLines
    ;
kStatLines
    :      kStatLine+
    ;
kStatLine
    :      (kALUsStat | kIfThenElseStat | kKOMDef)
    ;
```

4.2.1. Arithmetic Logic Units (ALUs)

Arithmetic Logic Unit is an atomic rule for checking. It returns true or false result by comparing left operand and right operand with an operator. Left operand describes a specific condition while the right one states an explicit value. With AND, OR conjunctions, multiple ALUs can be joined in expressive way. The definition of ALUs is as follows:

- [3-1] ALUs definition: kALUsStat

```
kALUsStat
    :      kALUStat ((AND | OR) kALUStat)*
    ;
kALUStat
    :      kLOperand kALUOperator kROperand
    ;
```

The `kALUsStat` defines multiple ALUs joined with conjunctions and the `kALUStat` defines a single ALU. The `kLOperand` and the `kROperand` can be predefined methods, dot notation access to building objects and properties, or explicit values such as truth or falsity, string, or numeric values. Operators are data type-specific. Operators defined by the `kALUOperator` can be as follows:

- Operators for string: =, !=, ==
- Operators for numeric: >, >=, =, <, <=, !=

Some examples of the valid KBimCode segments within the rule `kALUsStat` are as follows:

```
1) getSpaceDistance(LivingRoom, Stairs, MRP) <= 30;
2) Rail.Space.type = Balcony AND Rail.Floor.number >= 2;
3) getResult(BA.49.1) = TURE;
```

4.2.2. Conditional Statement

[3-2] Conditional Statement definition: kIfThenElseStat

```
kIfThenElseStat
    :      kIfStat kThenStat kElseIfStat* kElseStat*
    ;
```

The conditional statement inheres IF-THEN-ELSEIF-ELSE logic. The ELSEIF and ELSE are optional. Following are example snippets.

```
1) IF(BuildingStoriesCount()>=6) THEN isExist(Elevator)=TRUE
2) IF (CS) THEN KS
```

Logically, 1) and 2) are same. The inherent ALUs can be replaced with variable name of the Statements Group (in this example, CS and KS).

4.2.3. KBimCode Object Model

Korea Building Act has its own semantics of building objects and their properties which are different from that of IFC or proprietary BIM platform. KBimCode Object model (KOM) is a human-centred abstraction of building objects based on the semantics of the Korea Building Act. It is borrowed concept from (Lee, 2011). By using KOM, users can derive the objects of interest. KOM is dynamically instantiable with user-defined condition rules.

The basic syntactic form of defining KOM is as follows:

```
ObjectType var {
    Statement;
    …
}
```

– Where `ObjectType`, `name`, `Statement` are non-terminal tokens.

The non-terminal token `objectType` are class level objects defined in the KOM lexicon. The `var` is a user-defined variable name that can be replaced with the dynamically instantiable KOM. The `Statement` is KBimCode statements that specifies constraints of the KOM.

The definition of KOM is as follows:

[3-3] KOM definition: kKOMDef

```
kKOMDef
    :      kKOMDefStatDec
           '{' kStatDef '}'
    ;
kKOMDefStatDec
    :      bWrapObjType BID
    ;
```

The `bWrapObjType` defines type of building objects. The lexical rule `BID` can be instantiated by any of the variable names. The examples of KOM definition and application are described in the following evaluation section.

5. Evaluation

This section shows snippets of KBimCode. Enforcement Decree of Building Act (EDBA), Article 35, Clause 1 (NSDCK, 2015) was selected as an example. It regulates the installation of direct stairs. Table 1 shows the building act sentences and the corresponding KBimCode.

Table 1. Example of KBimCode based on actual Korea Building Act: "Enforcement Decree of Building Act Article 35, Clause 1".

Korea Building Act : Enforcement Decree of Building Act (EDBA), Article 35, Clause 1
[EDBA.35.1] Direct stairs installed on the fifth or upper floor or the second or lower underground floor shall be installed as fire escape stairs or special escape stairs: Provided, That the same shall not apply to cases where main structural parts are made of a fireproof structure or noncombustible materials and falls under any of the following subparagraphs:
KBimCode

```
check(EDBA.35.1) {
        IF !(CS1 AND CS2) THEN KS
}
CS1 {   isFireResistantStructure(MainStructure) = TRUE
        OR getObjectMaterialType(MainStructure) = Non-combustible;
}
CS2 {   getResult(EDBA.35.1.1) = TRUE
        OR getResult(EDBA.35.1.2) = TRUE;
}
KS {    Floor myFloor {
                myFloor.number >= 5
                OR myFloor.number <= -2;
        }
        Stair myStair {
                myStair = getObject(DirectStair);
                getObjectProperty(myStair) = EscapeStair
                OR getObjectProperty(myStair) = SpecialEscapeStair);
        }
        hasElement(myFloor , myStair) = TRUE;
}
```

Evaluation of KBimCode Language is performed using parsing tree which is auto-generated by a parser. Figure 2 shows part of the evaluation results: KBimCodeProgram, Check declaration, and KOM definition. KBimCode segments in the Table 1, the EDBA 35.1 was used as the sample KBimCode for the evaluation.

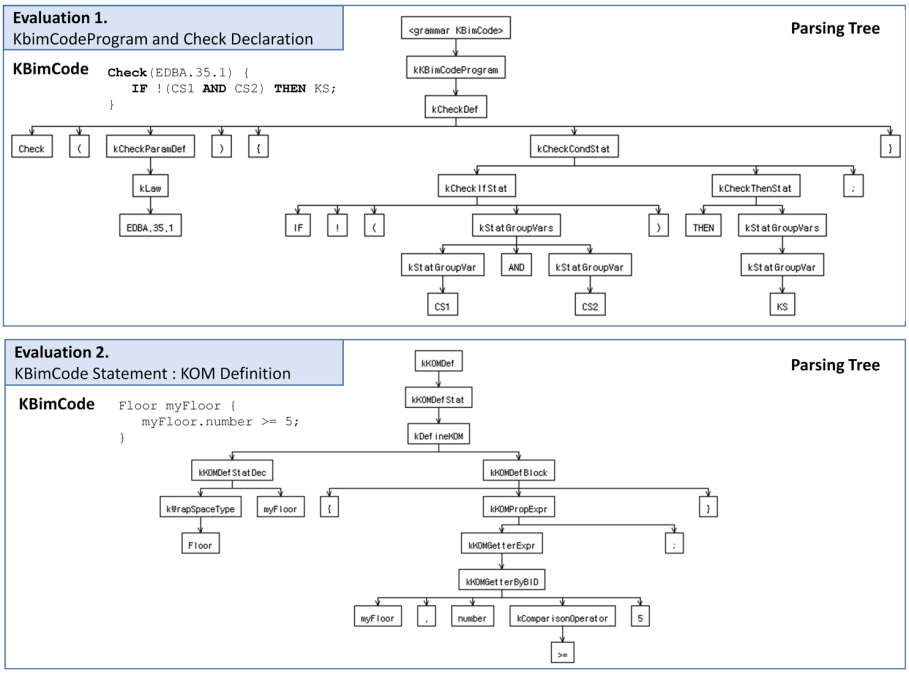

Figure 2. Snapshots of the evaluation results of KBimCode Language with parsing tree.

6. Summary

This paper introduced the approach to language design and definition of KBimCode. The lexical and syntactic form of KBimCode Language reflects features of building permit regulations in Korea Building Act and can be categorized as; 1) Check declaration, 2) Statement Group, and 3) KBimCode statements. The Check declaration specifies a target building act sentence to be checked and contains a set of statements that express conditions of checking. Types of statements are ALUs, conditional statement, and KOM. Especially, KOM is one of the key features of KBimCode. The set of statements can be grouped into the Statement group and used as a reusable object.

KBimCode development is an ongoing project. Further development of KBimCode Language will reflect extended syntax as KBimCode aims to extend its target to various requirements such as design guideline, request for proposal, etc.

Acknowledgements

This research was supported by a grant (15AUDP-C067809-03) from Architecture & Urban Development Research Program funded by Ministry of Land, Infrastructure and Transport of Korean government..

References

Ding L., Drogemuller, R., Rosenman, M., Marchant, D. and Gero, J.: 2006, Automating code checking for building designs, *CRC for Construction Innovation 2006*, Australia.

Eastman, C.: 2009, Automated assessment of early concept design, *Article in Architectural Design Special Issue: Closing the Gap*, **79**(2), 52–57

Eastman C., Lee, J.-m., Jeong, Y.-s. and Lee J.-K.: 2009, Automatic Rule-based Checking of Building Designs, *Automation in Construction*, **18**(8), 1011–1033.

ECMA: 2015., "ECMA-404 The JSON Data Interchange Standard". Available from: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> (accessed November 2015).

Han, C., Kunz, J. and Law, K.: 1997, Making automated building code checking a reality, *Facility Management Journal*, 22–28.

"Korea Building Act": 2015. Available from: National Statute Data Center Korea <http://elaw.klri.re.kr/kor_service/main.do> (accessed November 2015).

Lee, H.; Lee, S.; Park, S. and Lee, J.-K.: 2015, An Approach to Translate Korea Building Act into Computer-readable Form for Automated Design Assessment, *ISARC2015*, Oulu

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

Lee, J.-K., Lee, J., Jeong, Y.-s., Sheward, H., Sanguinetti, P., Abdelmohsen, S. and Eastman, C. M.: 2012, Development of space database for automated building design review systems, *Automation in Construction*, **24**, 203–212.

Malsane, S., Matthews, J., Lockley S., Love, P.E.D. and Greenwood D.: 2015, Development of an object model for automated compliance checking, *Automation in Construction*, **49**, 51–58.

Nawari, O. N.: 2012, Automated Code Checking in BIM Environment, *14th International Conference on Computing in Civil and Building Engineering*, Moscow.

Park, S.; Lee, H.; Lee, S.; Shin, J. and Lee. J.-K.: 2015, Rule Checking Method-centered Approach to Represent Building Permit Requirements, *ISARC 2015*, Oulu.

Parr: 2008, The Definition ANTLR Reference: "Building Domain-Specific Languages", Pragmatic Bookshelf.

W3C: 2015, "XML". Available from: <http://www.w3.org/TR/WD-xml-961114.html> (accessed November, 2015).

Wikipedia: 2015, "Definition of ALU". Available from: <https://en.wikipedia.org/wiki/Arithmetic_logic_unit> (accessed November 2015).