J Intell Robot Syst (2015) 79:507–522
DOI 10.1007/s10846-014-0117-7

Implementation of a BIM Domain-specific Language for the Building Environment Rule and Analysis

Jin-Kook Lee · Charles M. Eastman · Yong Cheol Lee

Received: 17 January 2014 / Accepted: 11 June 2014 / Published online: 21 September 2014

Abstract This paper describes an implementation process for a domain-specific computer programming language: the Building Environment Rule and Analysis (BERA) Language. As the growing area of Building Information Modeling (BIM), there has been a need to develop highly customized domain-specific languages for handling issues in building models in the architecture, engineering and construction (AEC) industry sector. The BERA Language, one of the domain-specific languages, deals with building information models in an intuitive way in order to ensure the quality of design and assess the design programming requirements using user-defined rules in the early design phases. To accomplish these goals, the BERA Language provides the capabilities for an effectiveness and ease of use without precise knowledge of general-purpose languages that are conventionally used in BIM software development. Furthermore, the design and implementation of the BERA Language focuses on building objects and their associated information-rich properties and relationships. This paper represents the implementation issues of the BERA Language associated with the building information models, their mapping into the building data structure, and their instantiation and execution. In addition, Portability of the language, extensibility and platform-dependent issues are involved in the BERA Language implementation. The implementation described in this paper is based on the use of Industry Foundation Classes (IFC) as given building information models, Solibri Model Checker® (SMC) as an IFC engine, and the Java Virtual Machine (JVM) as a compilation and execution environment.

Keywords BERA Language · Language Implementation · BIM · IFC · Design Rule Checking

J.-K. Lee (✉)
Department of Interior Architecture Design,
Hanyang University, Seoul, Republic of Korea
e-mail: designit@hanyang.ac.kr

C. M. Eastman · Y. C. Lee
Digital Building Laboratory, College of Architecture,
Georgia Institute of Technology, Atlanta, GA, USA

C. M. Eastman
e-mail: chuck.eastman@coa.gatech.edu

Y. C. Lee
e-mail: zsradox0@gmail.com

1 Introduction

This paper aims to describe an implementation process for a domain-specific computer programming language: the Building Environment Rule and Analysis (BERA) Language. The advent of computer applications supporting the three dimensional object modeling of buildings, called Building Information Modeling (BIM), allows both automatic parametric generation of designs that respond to various criteria and the prospect of computer-interpretable models

and automated checking of designs after they are generated [1–4]. In addition, this emerging flow of BIM has led to develop highly customized domain-specific languages for handling issues in building models in the architecture, engineering and construction (AEC) industry sectors. This paper depicts one of the domain-specific languages, the BERA Language, which attempts to deal with building information models in an intuitive way. The application of the BERA Language aims to provide efficiency in defining, analyzing and checking rules.

In addition, this paper represents an implementation process of the BERA language architecture supporting ease of use, high fidelity, extensibility and portability. The purpose of the BERA Language enables users to accomplish an effectiveness and ease of use without precise knowledge of general-purpose languages conventionally used in BIM software development. To achieve these goals, this study proposes an abstraction of the universe of discourse - the BERA Object Model (BOM). BOM plays a pivotal role as baseline objects for the implementation of the BERA Language. The implementation issues are described with the building information models, their mapping into the building data structure, and their instantiation and execution, focusing on building objects and their associated information-rich properties and relationships. Portability of the language, extensibility and platform-dependent issues are also involved in the BERA Language implementation. The implementation is based on the use of Industry Foundation Classes (IFC) as given building information models, Solibri Model Checker® (SMC) as an IFC engine, and the Java Virtual Machine (JVM) as a compilation and execution environment. For efficient development of the language, this paper takes advantages of some useful reference languages such as ECMA Script language and Java-based domain specific languages in terms of their syntax style and execution pipeline

2 Background

2.1 Scripting Languages

Although their boundaries are often too ambiguous to differentiate, computer programming languages can be classified based on several perspectives such as their chronology, category, generation, paradigm, target user, standardization, and so on. A standard language can even have many dialects for different purposes. This study introduces two sets of languages and represents what features can be borrowed to use in the design of the BERA Language. As a brief introduction, the first set contains JavaScript and ActionScript scripting languages in order to review their effectiveness and scalability according to the expansion of target objects. The second set is the Processing language that is one of the Java-based domain-specific languages in order to speculate how it is designed to alleviate difficulties for both users and developers. These languages are similar to each other in certain points because they are created to shorten the conventional language execution process such as edit – compile – run cycle [5]. Especially the scripting languages aim to support software applications quicker and easier by end-users.

ECMAScript is a widely used scripting language especially for the World Wide Web. It is commonly used in well-known dialects of JavaScript and ActionScript. It has been standardized by ECMA International in the ECMA-262 specification and ISO/IEC 16262 in 1997 [6–8]. Relying on World Wide Web Consortium (W3C) [9] supported various Web standards and APIs, ECMAScript became one of the most popular scripting languages to users. JavaScript is supported in many common web browsers for handling client-side web documents. Most HTML [10]-based web page developers are familiar with JavaScript and know how to handle web-document elements such as text field, image, button, selection box, and other various web-forms. JavaScript recently has been more and more popular because there are many existing standard-based new ways for developing web interfaces within the Web 2.0 [11], such as XHTML [12, 13], CSS [10], XML [14], AJAX [15], jQuery library [16], and so on.

JavaScript was introduced in 1996 through the Netscape web-browser [17]. Due to the success of the World Wide Web, JavaScript has been popular to people even to those who are not familiar with computer languages. It is an object-oriented scripting language that runs on client-side web browsers to access objects within applications. It is influenced by many languages and its syntax is similar to the Java language as its name implies, and so it is easier for novice programmers to develop dynamic web pages. It enables the control of elements in web pages such

as W3C's document object model (DOM) [18], and makes user interfaces more interactive. Figure 1 shows a simplified overview of DOM. It allows for accessing a page document and controlling its elements by means of scripts. JavaScript uses dot-notations to access not only the document's associated objects and properties but also the function calls in an intuitive way. This is also another important distinct feature of object-oriented concepts and implementation. An instantiated object has its properties as well as its pre-defined behavior (e.g. methods, function calls). Many recent object-oriented commercial programming languages commonly accept it simply because it is faster to write and clearer to read [19]. For instance, based on the DOM, a navigation history can be accessed by the dot-notation: `window.history`, and a button in a form can be accessed by the notation: `window.document.forms.button`.

Another famous dialect of ECMAScripting language is ActionScript. It has also been broadly used mostly on the web, embedded in Flash movie clips [20]. As Adobe Flash™ (formerly Macromedia Flash) [21] has been used increasingly on the web, ActionScript became more popular in producing sophisticated movies.

As JavaScript controls web DOM, ActionScript controls the frames and behavior of the movie. They are very effective and easy to handle specific problems in each target domain. As the nature of the scripting language and domain-specific language implies, ECMAScript-based domain-specific languages influence many features for the development of the BERA Language, especially concerning their abstraction principle and means of handling target objects. BERA attempts to handle a pre-defined complicated model – the building information model. A building information model, such as one implemented in IFC, already provides a well-structured data format in its own scheme, but its internal data structure is usually complicated and heavy for casual users. In BIM software, building data structure is represented in an explicit way, but users tend to approach it in an implicit way because of the nature of a building design. ECMAScript stands as a precedent since it supports a pre-defined and standardized object model based on domain-centred user-defined names. The way target objects (DOM or Flash Movie) are managed by such scripting languages is not so difficult for domain experts. In JavaScript for example, users are accustomed to doing their job with target object models at a high-level. They do not need to comprehend how document models, client browsers, network protocols, and web servers are computationally inter-operated with each other at a low-level. This DOM concept is primarily used to generate BOM, which allows users to manage object-based and domain-specific information.

2.2 Domain-Specific Languages Based on JAVA

A domain-specific language is usually hard to distinguish between small-sized general programming languages and scripting languages such as JavaScript. However, they are commonly found in modern

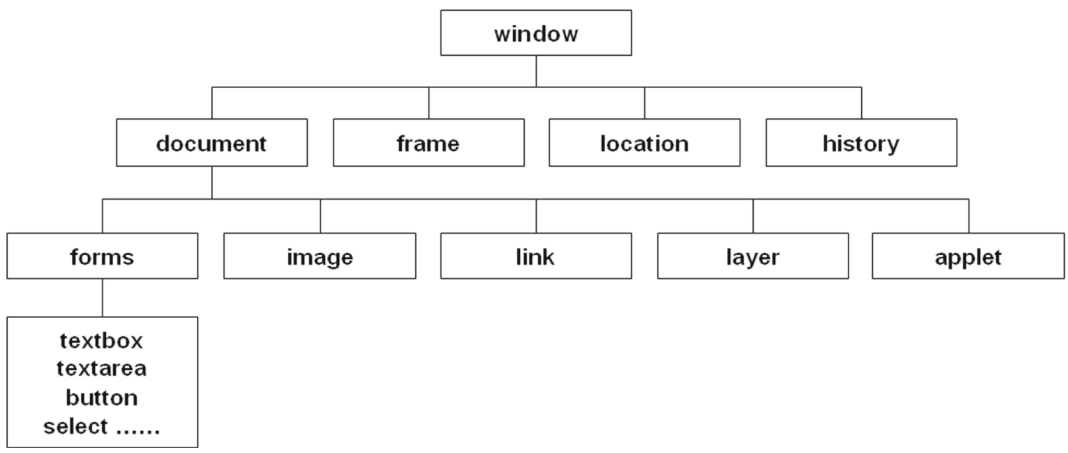

Fig. 1 Example of the target objects of JavaScript: document object model (DOM)

computing environment. Examples include HTML embedded in web programming languages, CSS [22] in web scripting languages, Regular Expressions [23] embedded in many other programming languages, and so on. Domain specific languages typically have the connotation of being smaller due to their specific purposes.

The architecture of Java-based domain-specific language has its influence on the development of the BERA Language in some ways, especially in actual implementation. However, the BERA Language has another huge layer of implementation: the building model platform and its bridges to the BERA Language. IFC is a reasonable target building model because it is an effort to normalize the various native building models embedded in BIM platforms. Current BIM platforms have the translators that support IFC import and export, and IFC is a close approximation to platform specific native building models. In the scope of this study, therefore the building model is in IFC format, and it basically transfers BIM data to be consumed in the framework of BERA Language. Accordingly, a simpler and neutral building model that is available to users is necessary. This model should be easily accessible to users instead of accessing either platform-dependent or usually very complicated native data structure including IFC. Moreover, it should still provide high fidelity to be useful on the specific problem domains even if it is simplified and neutralized.

3 BERA Language Design

One of the important factors to be considered in developing domain-specific languages is to provide high fidelity to problems. Not only is the capability of addressing new domain-specific problems necessary, but also the substantial ability to raise the level of abstraction on target elements is strongly required [24, 25]. The BERA Language has a strong bond to the building information model such as IFC. A given building model is always defined within its own data structure: native and partially-open data structures from native building models, open and neutral data structure from IFC, etc. [26] describes how to build an abstract model from these existing building models, especially from IFC, named BERA Object Model (BOM). BOM is a semantic set of data translated and parsed by the predefined schematic rules. BOM stores structure and information of a building model and allows the executor to efficiently analyze and implement it. Thus, we can say even though IFC and BOM have same information for building models, the building data in the latter format is more applicable and executable for rule-based checking than the former one. The BERA Language is based on two types BOM – static and dynamic. Static BOM is a static data set from a specific given building model, and can be represented by class names such as Building, Floor, and Space as discussed in the language design chapter. The Static BOM is mostly pre-determined by the given building model, and therefore most of the given property values are statically established when the building model is loaded into memory. Some properties can be both statically and dynamically assigned by user inputs for further development. For example, "buildingType" under Building object can be assigned by users using Handler_bReference (See Fig. 2), and "security" under Space object can be assigned by additional BERA library which is in charge of automated assignment of security level, even if their default values are empty. This is an example of the technical BERA Language extensibility for further use cases. The definition of BOM is open-ended, and the authors realized that it is another challenge to define generic and valuable BOM as it grows more detailed.

Abstraction is an important process in language design, because some of the language keyword tokens and syntaxes will be derived from this model. In other words, this simple-yet-implicit abstraction will be used by front-end users, but the complicated-yet-explicit IFC data structure (or a host native object model) still needs to be accessed and used in the implementation stage of the BERA Language.

The syntax of BERA Language is structured similarly to most programming languages such as Java and ECMAScripting languages. Each declared language element in BERA has a close relation with corresponding entities or properties in the given building model. In addition, the rules are dealing with as many properties as the BOM can provide. As BERA Language is running on top of a given building model, BOM definition is the first step, and then rule definition will be followed using the declared objects. These object declaration or rule definitions can be derived from the external dataset

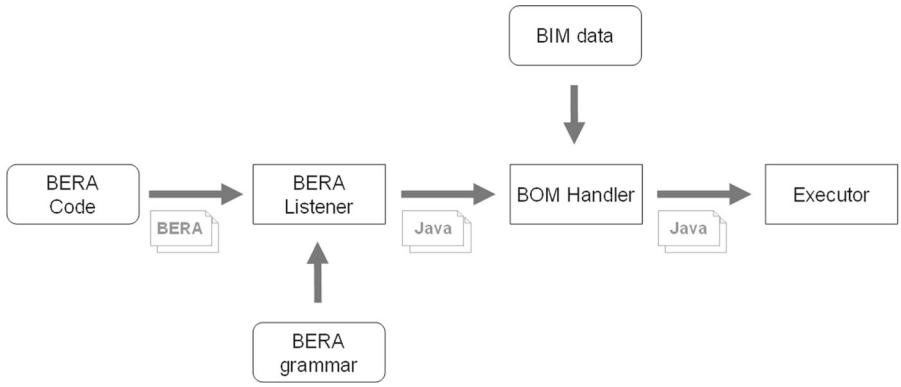

Fig. 2 A brief data flow diagram to describe the implementation of the BERA language tool. A BERA code is translated into a Java code and executed

such as pre-defined library and external BERA program file, thus, "import" functionality is also required at the beginning of the BERA program statement. In the highest level, a BERA program has four components as follows:

- BERA reference directive: `bReference`
- BERA Object Model definition and declaration: `bBOMDef`
- BERA rule definition: `bRuleDef`
- BERA execution statement: `bExeStat`

The term bReference, bBOMDef, bRuleDef, and bExeStat will be represented in following implementation section in detail, and sub-divided into their sub-components level for describing the language implementation process. Detailed language design issues and definitions are described in [27].

4 BERA Language Implementation

4.1 Implementation Overview

The BERA Language has been designed and defined so that its implementation is portable to different building information modeling platforms. Figure 3 describes the execution architecture of the BERA Language. In the BERA Language implementation stage, two main environments should be covered: the platform-free environment (in other words, the BERA Language-specific and common front-end part for different platforms) and the BIM platform-dependent environment. The front-end part is standard for all other implementations, while the back-end part varies by BIM platform. The basis of the target implementation of this study is the Industry Foundation Classes (IFC) as given building information models, SMC as an IFC engine, and the Java Virtual Machine (JVM) as a compilation and execution environment.

Regarding implementation, the BERA Language architecture consists of two high-level components as described below.

- Front-end: BERA engine: This contains user-generated language programs, the BERA translator/interpreter to the target language, and other intermediate representations and executors for generating the BERA Object Model. This front-end engine is standard for all implementations and environments.
- Back-end: Custom engine: The BERA Language could not be executed without a given building model. The building information modeling engine is another huge platform. SMC is a candidate platform that the BERA engine uses, both in terms of object models and applications. This back-end implementation varies by platforms, but the target intermediate model is always the BERA Object Model.

This chapter describes one of the BERA Language implementation approaches regarding a plausible structure for the general language back-end issues: lexical analysis, syntactic parsing, semantic analysis, intermediate code generation, target code generation, and execution [28]. The back-end side issues and their implementation can be flexibly adopted by developers. The implementation approach described in this chapter is based on the actual application named the BERA Language Tool version 1.0. This chapter focuses on generic issues of the BERA Language Tool rather than implementation and platform-specific details.

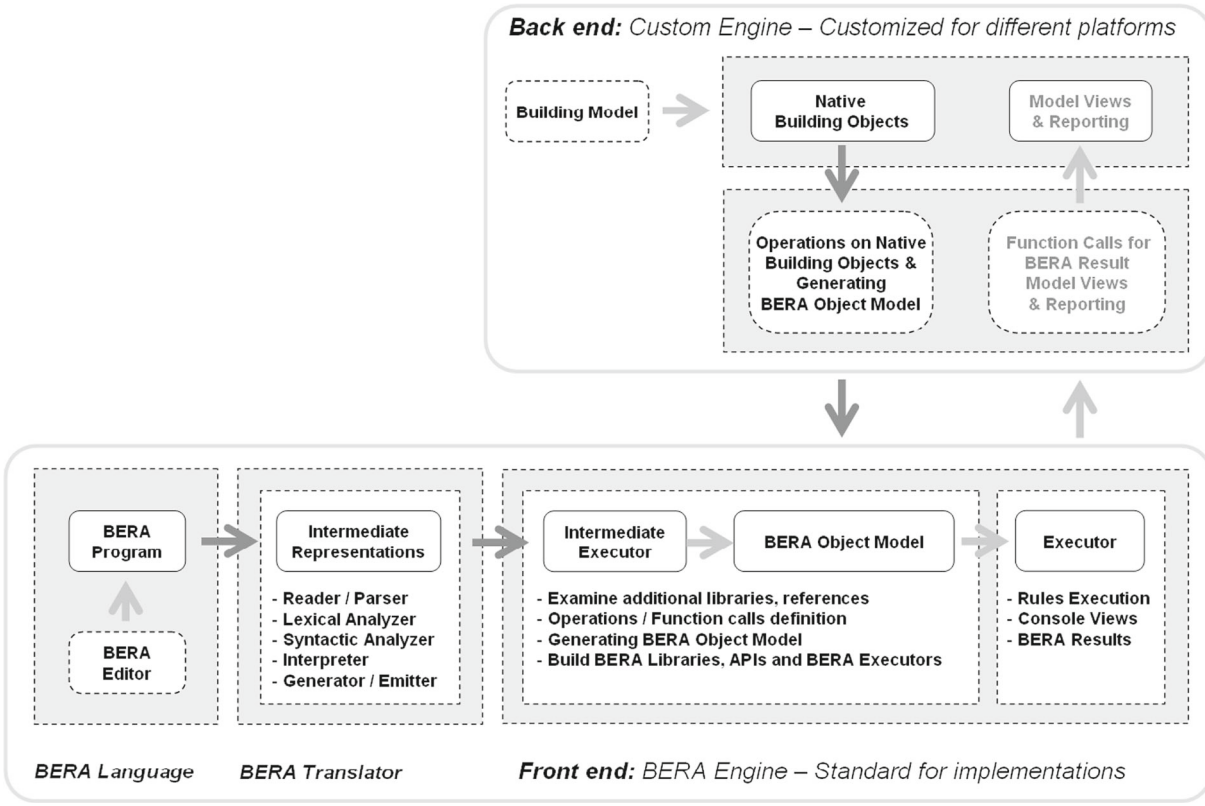

Fig. 3 BERA language architecture detail: front-end and back-end

4.2 BERA Listener

4.2.1 Lexical and Syntactic Analysis

User textual input stream should be parsed and recognized before establishing its semantics and execution. The module named BERA listener is in charge of this first process of language recognition – lexical and syntactic analysis. Language recognition is a very important step in any language application. Although this technical parsing has an important role in language implementation, it is beyond the scope of this work. This paper does not attempt to tackle general and detailed issues concerning parsing or its patterns. Some of the parsing modules in the BERA Language Tool have been facilitated by ANTLR [25, 29–32] as a parser generator for user BERA Language input, especially for the lexical and syntactic analysis steps – the first stage of the BERA Language recognition. Instead, the authors briefly introduce how these input texts can be handled in terms of building BERA semantics. The series of input tokens will be classified by the BERA listener as shown in Fig. 4.

Figure 4 shows four components that are in charge of handling input texts. They are equivalent to the highest classification of the BERA programs: bReference – DefRef, bBOMDef – DefBOM, bRuleDef – DefRule, and bExeStat - DefExe. Input texts are still considered unknown textual stream to computers. However, by using the BERA listener, they can be classified according to the high-level BERA Language structure and definition:

DefRef (Reference Declaration),

DefBOM (Dynamic BOM Declaration),

DefRule (Rule Definition Statement), and

DefExe (Execution Statement).

Lexical and syntactic analyzers in the BERA listener play important roles to validate, report, and

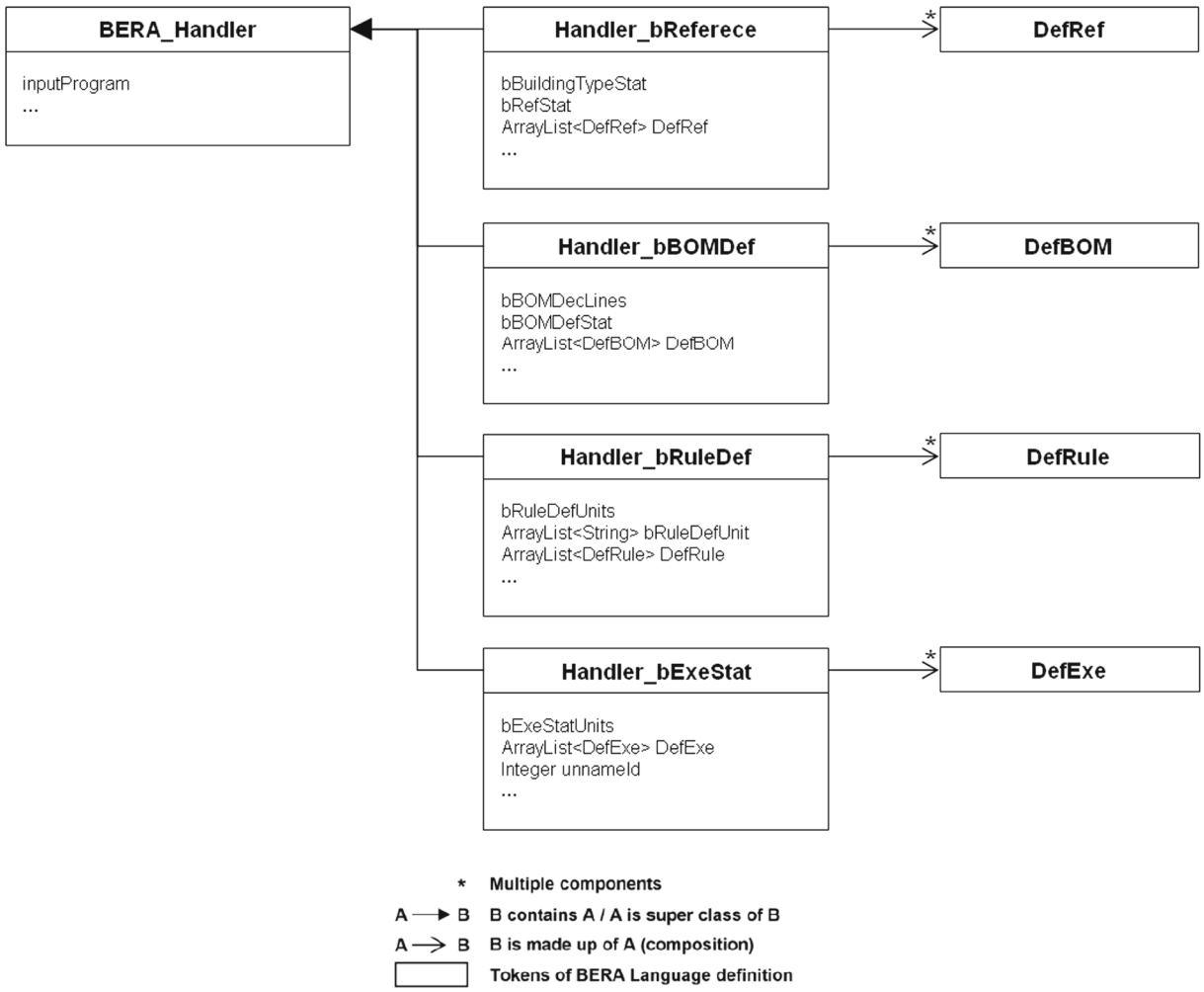

Fig. 4 An overview diagram to describe the top-level lexical and syntactic analysis of BERA Language input program

classify the user-input texts for the subsequent and more important task: semantic analysis.

4.2.2 Semantic Analysis

For the current stage of preliminary study, the demonstration of material dispatching is not performed. The uncertainty events of operation re-planning are simulated in the study. And the accuracy and stability evaluations of tracking devices for operation guidance re-planning are under investigations. The sensing abilities for establishing a situation awareness environment should be exclusively addressed in the future, for the whole processes of LNG modular construction.

As Fig. 5 describes their hierarchy, DefBOM, DefRule, and DefExe can populate multiple DefBOM, and a DefBOM and a DefRule include DefCond which is the condition definition statements. A DefCond contains a single operand for a certain condition using a dot-notation access to BOM, operator, and value. An important aspect of this DefCond is that it also has a one-to-one relation to DOTExpr which defines its semantic meaning of the dot-notation access to BOM. The execution statement DefExe has a DefResult to instantiate its execution result. The BERA listener works out the implications of these input components that are validated and takes the proper executions. For example, a single DefCond can be derived from a DefBOM or a DefRule. The following is an example of a DefCond:

$$\text{path.one.distance} <= 100$$

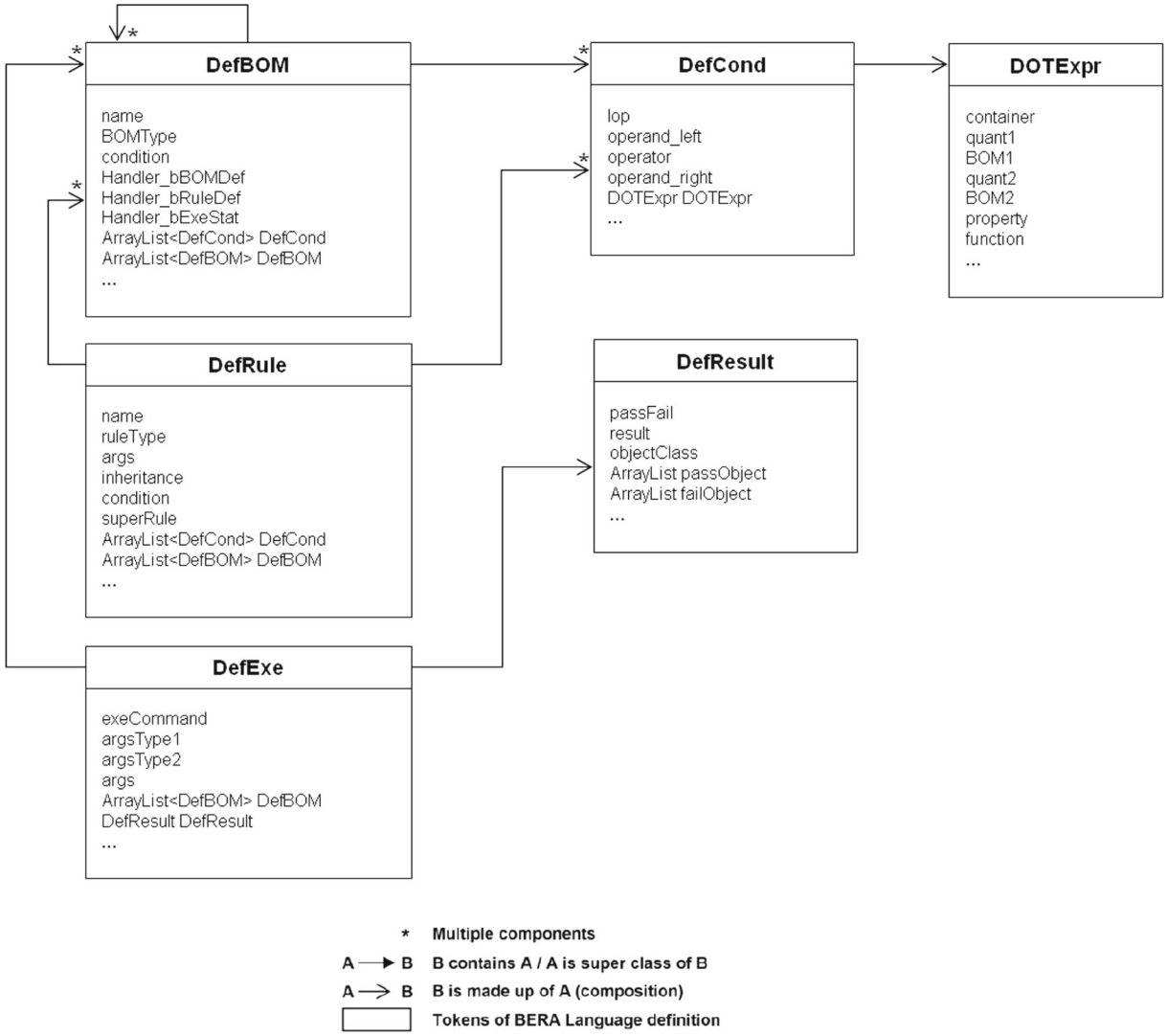

Fig. 5 An overview diagram to describe establishing the BERA Language semantics

This example DefCond can be derived from a DefBOM as one of the conditional statements to define an instance of SpaceGroup, or from a DefRule as one of the rule conditions defined as a rule. The DOTExpr instance populated from this DefCond can be represented as follows:

(container.)((quant1.)BOM1.)((quant2.)BOM2.)(property)

- Where all are non-terminal and optional tokens: For example, "path.one.Space.area" can be matched as follows: container – path, quant1 – one, BOM1 – Space, and property – area. (See Table 1 for more examples)

The semantic analyzer in the BERA listener is in charge of converting the text inputs that are processed by the lexical & syntactic analyzer into the object instances to be consumed in the language execution. This also contains most of the features of the semantic analysis such as BOM data type checking, the assignment of names or variables, and object binding. Table 1 describes the DOTExpr parser which has the most important role in establishing BOM semantics. The semantic analysis can be done by early syntactic level or late intermediate representation/execution level, considering the implementation environment. The use cases and examples are introduced in [26].

Table 1 Some valid examples of the dot-notation access to BOM (examples of DOTExpr in Fig. 5)

Tokens	DOTExpr Tokens	Example Expression
1	container	Space
	container	myOffices
2	BOM1.property	Space.area
	container.property	myPaths.distance
3	BOM1.BOM2.property	Space.Floor.name
	container.BOM1.property	myOffices.Space.height
4	container.BOM1.BOM2.property	myOffice.Space.Floor.height
	container.quant1.BOM1.property	myPath.one.Space.name
5	container.quant1.BOM1.BOM2.property	path.one.Space.Floor.height
	container.BOM1.quant2.BOM2.property	path.Space.one.Floor.height

In the example expression column, words starting with upper case letters are BOM names, and words starting with lower cases are property names or quantification words. User-defined variable names for dynamic BOM are in italic.

4.3 BERA Object Model Handler

The BERA Language is based on two types of BERA Object Model (BOM) – static and dynamic. Static BOM is a static data set from a specific given building model, and can be represented by class names such as Building, Floor, and Space as discussed in the language design chapter. The Static BOM is mostly predetermined by the given building model, and therefore most of the given property values are statically established when the building model is loaded into memory. Some properties can be both statically and dynamically assigned by user inputs for further development. For example, "buildingType" under Building object can be assigned by users using Handler_bReference (See Fig. 4), and "security" under Space object can be assigned by additional BERA library which is in charge of automated assignment of security level, even if their default values are empty. This is an example of the technical BERA Language extensibility for further use cases.

BOM builders are in charge of handling building objects and their properties. The static BOM builder is building model-specific; therefore there will be more emphasis in this chapter on describing the dynamic BOM builder. The focus of this study and implementation is on the spatial BOM such as SpaceGroup and Path, but the structural BOM instances (e.g., a sub class of ObjectGroup) are also instantiated and used in the tool because they have physical relations with spatial BOM instances. Figure 6 illustrates the BOM classes in the implementation described in this section. Briefly, the arrows between SpaceGroup, Path and others mean inheritance (the same as between Structure and others), and other arrows mean their association relation.

4.3.1 Static BOM Builder

The BOM handler establishes static BOM data when the model is imported, or before the user input language is parsed or executed. BOM is an abstraction of the complex building state focusing on its several "rule and analysis" perspectives. BOM is one of the key concepts to the building environment rule and analysis as the language name literally implies. In the BERA Language Tool implementation, many computed and derived properties have been proposed and implemented for specific purposes, as well as some basic data obtained directly from the given building model. The implementation takes advantages of the input building information, but additional implementation is required to compute some BERA-specific properties. These are managed by the static BOM builder.

4.3.2 Dynamic BOM Builder

The dynamic BOM builder is in charge of establishing user-defined collection of static BOM. These will be consumed in the design rule & analysis tasks, as is the case with static BOM. For example, in a circulation rule "circulation between A and B should be public", how can A and B be obtained? General rule-checking software uses space names to acquire them from a given building model. The BERA Language Tool can also support that, but in addition can provide a variety of sophisticated methods using this dynamic BOM

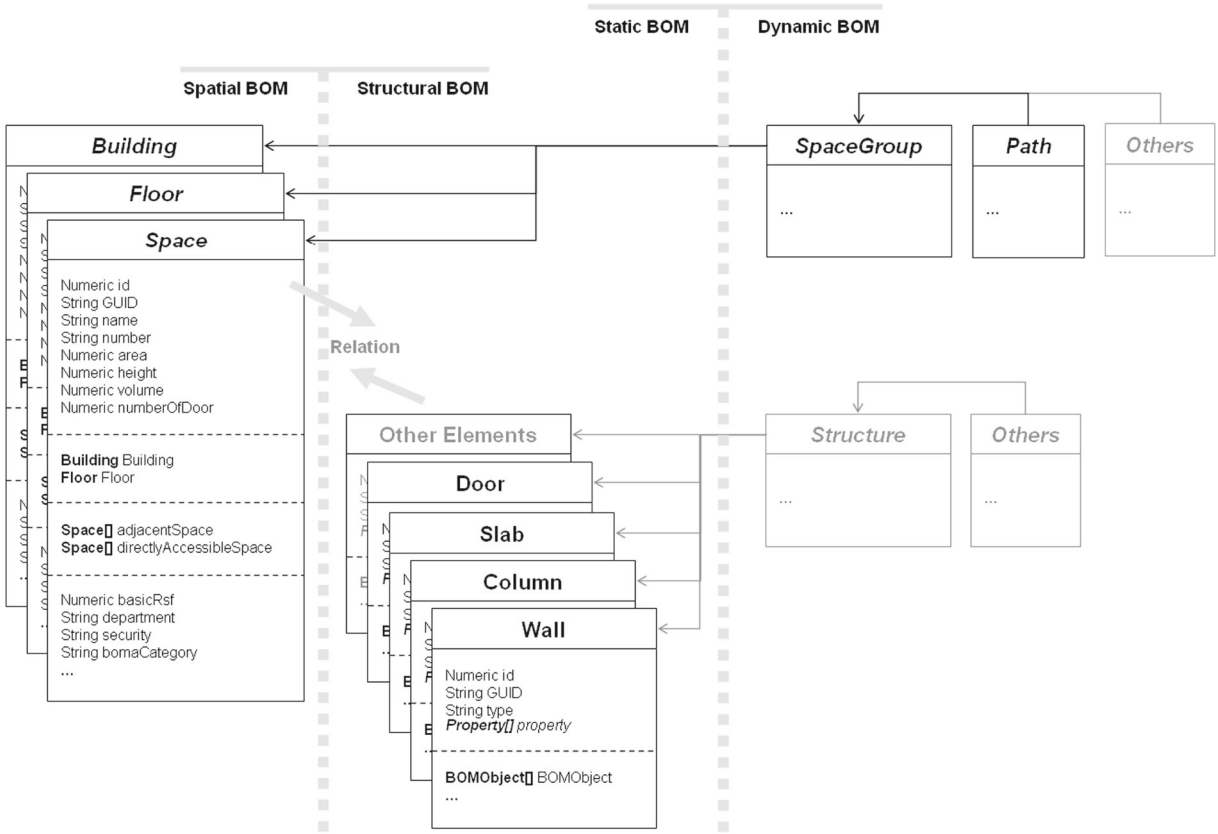

Fig. 6 Overview of implementation-level BERA object model classes

definition by users. Not only can their space names be applicable to obtain certain space collections, but also their spatial properties and relations.

This section introduces two major types of dynamic BOM implementation – SpaceGroup and another important sub class of SpaceGroup – Path. They can be originated from the super class ObjectGroup. The main difference between the dynamic BOM and the static BOM is its unlimited instantiability. Users can create any one of the user-defined SpaceGroup or Path instances using pre-determined static BOM in a given building model. Figure 7 describes those two classes and their super class – ObjectGroup. For handling and computing the building circulation-specific properties, there is another static meta-element – Graph. It is used in calculating metric distances and number of turns on the path, and these properties are stored under each Path instance. SpaceGroup is the super class of Path; therefore Path instances have all properties of SpaceGroup. For example, a Path instance also has properties of SpaceGroup such as 'numberOfSpace', 'height' and 'area'.

The dynamic BOM objects are instantiated by users' BERA Language (The objects named Def-BOM in Figs. 4 and 5). The dynamic BOM builder is in charge of instantiating those objects responding to users' ObjectGroup and object definitions. Any dynamic BOM is essentially a derived subset of static BOM. The user's variable name for a SpaceGroup will be a new container for that subset of static BOM. For example, a user-defined ObjectGroup "myOffices" can contain Space objects which are named "office", and this "myOffices" is the container for selected space objects. All the information is loaded in the structure DefBOM – DefCond – DOTExpr structure as shown in Fig. 5. One of the important features in the dynamic BOM builder is this kind of object selection algorithm as introduced in Fig. 8. This returns a series of Boolean results to determine whether the current element Object[n] could be selected or not for a given

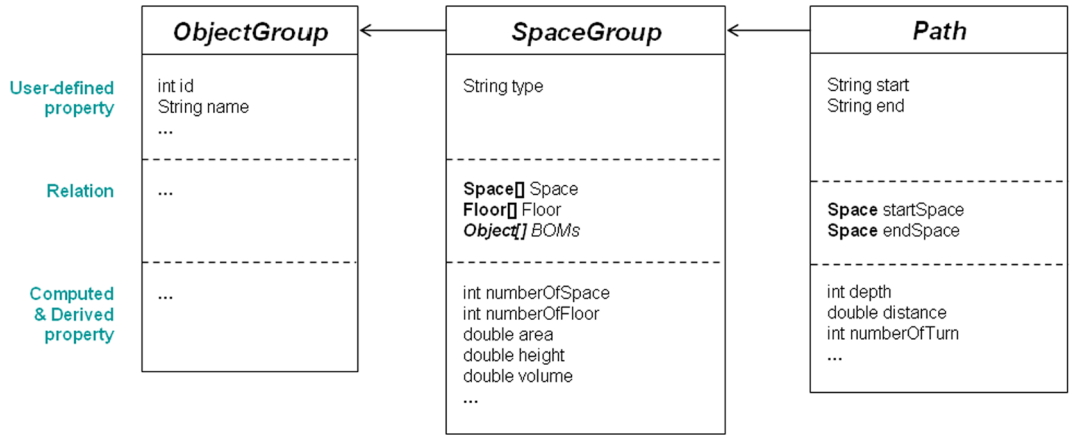

Fig. 7 Overview of the dynamic BOM and its properties implementation

DefBOM. This is also useful to execute the rules. As an example, Fig. 9 shows this process for the following dynamic BOM definition:

```
Space myOffice {
        Space.name = ''office'';
        Space.area > 500;
        or Space.Floor.height > 16;
        or Space.department = ''office'';
}
```

4.4 BERA Executor

The BERA Language execution statement basically consists of a simple line form – execution commands and their arguments. The fundamental command keyword is 'get', as described in the BERA Language design chapter. As it literally means, 'get' command retrieves all the BOM and visualizes them based on its arguments. This section focuses on rule checking – in this case, command keywords are user-defined rule names.

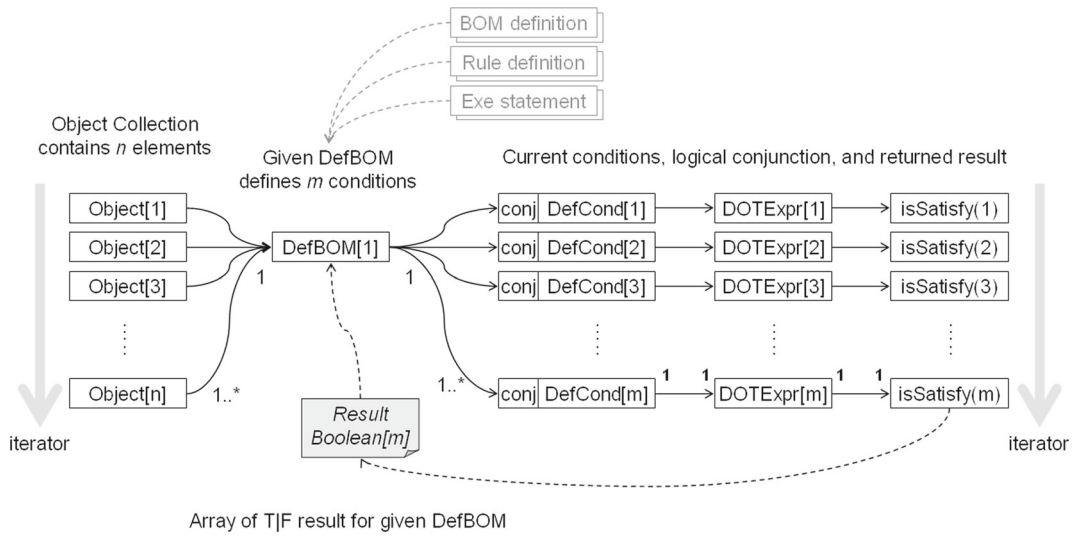

Fig. 8 Object selection algorithm overview: multiple conditions DefCond in a DefBOM iterate object collection and collect its array of Boolean results

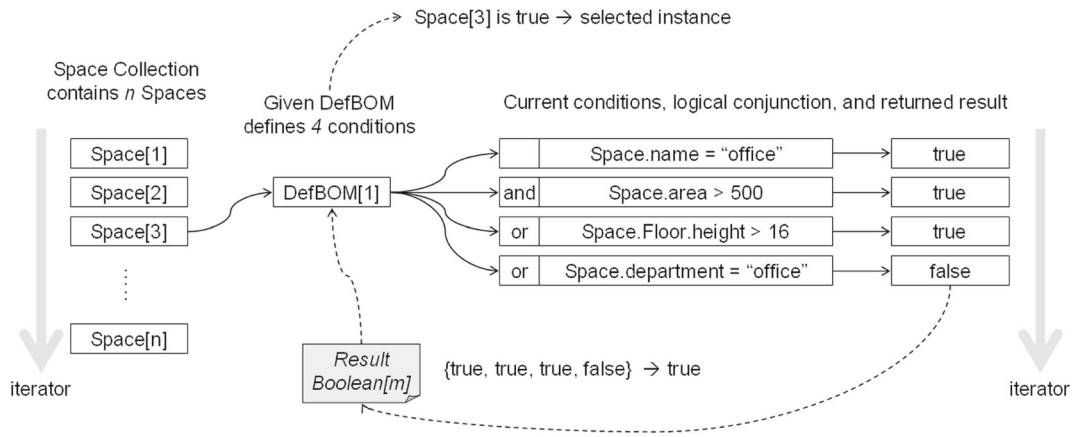

Fig. 9 Space object selection example for the above program: an instance Space[3] is selected because its result is true. A Boolean array {T, T, T, F} returns T because one of the conjunctions is "or" and its value is T. (*Left* to *right* evaluation)

User-defined rules basically consist of a variable name, a series of DefCond, and optional nested DefBOM as shown in Fig. 5. In the implementation level, DefRule and DefBOM have almost the same structure because they are eventually handled by DOTExpr representations and their object selection processes for either defining objects or regulating rule conditions.

The main difference between the definition of BOM and Rule is their container – a dynamic BOM definition basically has default or static BOM, but a rule definition has a dynamic BOM as its container. Similar to the series of examples to describe the object selection algorithm overview, Fig. 10 shows the object selection process as a rule checking process that is derived from a DefRule. A circulation path collection "p" is the container for this rule definition DefRule, and its DefCond emits the Boolean results through the iteration. This process occurs in the rule execution. A rule execution statement has a certain rule name as a function call, and it delivers user-variable arguments as given object containers. In this example, the container "p" contains n number of path instances and this process returns a series of Boolean results to determine whether this instance Path[n] is satisfied (selected) by the conditions or not. In the rule checking process, this selected instance means "passed" instance. The BERA executor is in charge of handling the process as well as considering given logical conjunctions on each condition. The simplest result of the rule checking execution is a Boolean – pass or fail; however, the BERA Language Tool provides an entire set of information gathered in this process to users.

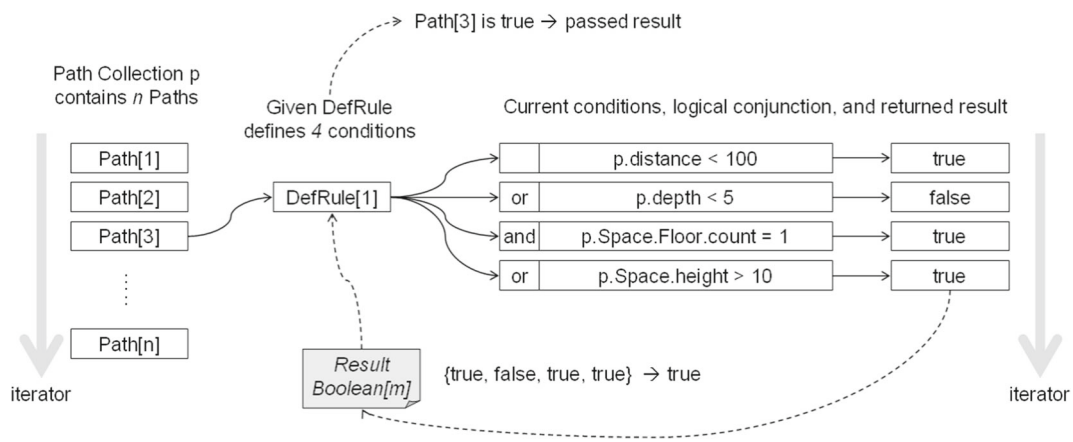

Fig. 10 Path object selection example: an instance Path[3] is passed because its result is true. A Boolean array {T, F, T, T} returns T because one of the conjunctions is "or" and its value is T. (*Left* to *right* evaluation)

4.5 BERA Language Tool

The BERA Language Tool is implemented as an integrated development environment of the proposed BERA Language. It is developed as a plug-in software on top of SMC, and runs on the JVM environment (See Fig. 11 as an example).

4.6 BERA Language Extensibility

Similar to the development of other programming languages, the BERA Language development is an open-ended project. Language syntax is technically the main subject of update. This section however emphasizes language extensibility issues focusing on its semantics – front and back-end extensibility. There are two different directions of the BERA Language extensibility:

Back-end extensibility: Re-targetable BERA Language to support other types of BIM platforms such as BIM authoring tools, model checking tools or simulation tools. For example, currently BERA is designed to operate in the platform of SMC. However, since BERA is one of languages, if other BIM authoring tools such as Revit Architecture or Tekla have features that accept the implementation and structure of BERA language, BERA would be successfully executable at these platforms as well.

Front-end extensibility: Extensible BERA Object Model, as well as the issues of BERA Language syntactic/semantic improvement, upgraded BERA Language Tool, etc. BOM extensibility is twofold: lateral extensibility (more building elements responding to the demand of new rules) and vertical extensibility (properties of elements). BERA language embedded in SMC platform generally supports the checking types executed by validation algorithms that SMC provides. Users can validate their IFC instance files with regards to building circulation, spatial programming, and specific query checking. For the circulation analysis, a SMC platform dynamically computes paths based on the defined BERA language. The graph structure represents the calculated walking distances and paths based on a length-weighted graph structure for a proposed building model.

This type of extensibility has been researched and executed for other types of rules such as accessibility and visibility for a hospital design checking. This new

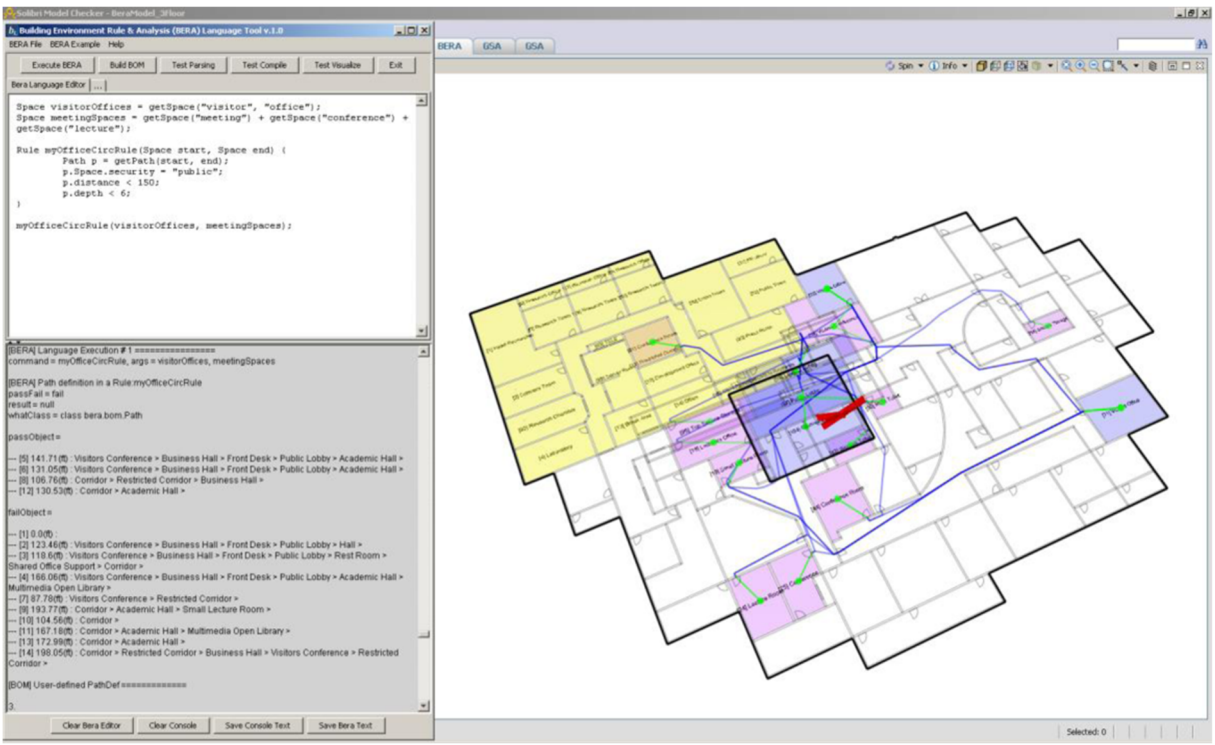

Fig. 11 BERA Language Tool on *top* of the BIM platform – SMC

project proved the capability of BERA language that can extend the areas of rules.

4.6.1 Re-Targetable BERA Language

User BERA Language programs are translated by the reader, and then the interpreter performs a series of internal processes such as lexical, syntactic and semantic analysis, data collection, generation of intermediate representations, and so on. As a result of the translation, a BERA program is re-generated in the target language internally. The series of processes by the BERA listener and BOM handler make the users' input language executable. Issues on the back-end extensibility arise in this phase because they are platform-dependent. How can the BERA Language be transplanted to different platforms? From a software engineering standpoint, the principle of "separation of concerns" [29, 33] may give clues to the BERA Language back-end extensibility.

In the actual implementation of the BERA Language Tool, one of the main concerns is that the BERA Language aims to be a re-targetable language considering its extensible capability to other platforms that are developed by different languages and libraries. The target language in this implementation is Java, but other general-purpose languages such as C++ and C# are also available for application using model-driven language translation engines [29, 31, 34, 35]. This approach makes the BERA Language re-targetable to other platforms. For advanced users, the BERA Language Tool also supports its target language directly from the BERA editor, and this enables users to handle a very detailed level of data as well as the API of the target BIM platform.

4.6.2 Extensible BERA Object Model

The definition of BOM is open-ended, and the authors realized that it is another challenge to define generic and valuable BOM as it grows more detailed. The current BERA Language focuses on the applications of evaluating building circulation and spatial programming with respect to the scope of the research and implementation. The dot-notation access to BOM is intuitively used in both the definition of BOM and rules. There are also several structural building elements available in the current implementation, as shown in Fig. 12. These have direct relations to the spatial objects such as spaces and floors, and they are instantiable in the current BOM. For example, Structure is the dynamic BOM similar to SpaceGroup or Path as another sub-type of ObjectGroup. As these are beyond the scope of this paper, their properties are not deeply developed yet – some default properties are available directly from the building model. However, they are still building elements that have direct relations to spatial objects. For example, a dot-notation operand mySpace.Door.width returns one or many numeric values of the width of the doors in a

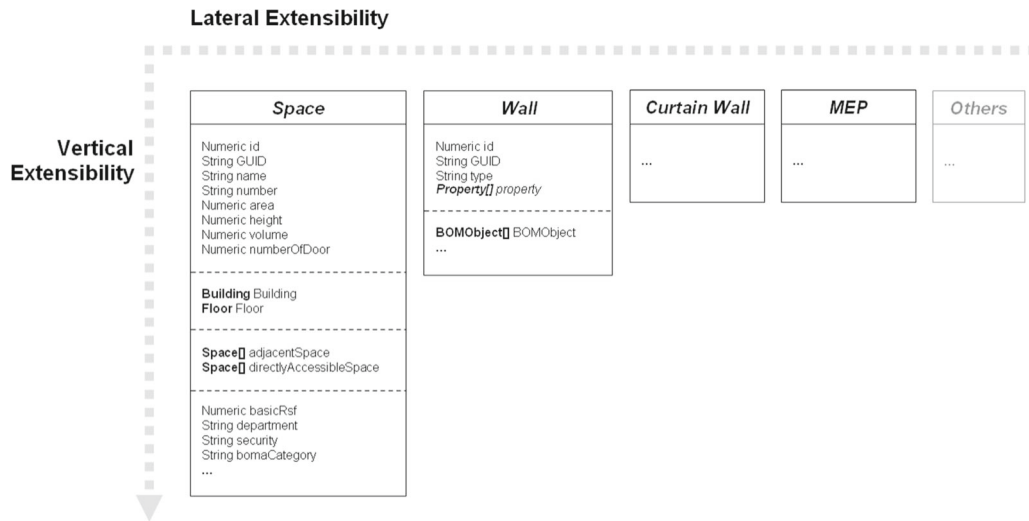

Fig. 12 Two-way extensibility of the BERA object model

group of spaces named mySpace, as it implies. Lateral extensions such as structural building elements and vertical extensions (such as additional properties for existing BOM objects) are good examples of the open-endedness of the BERA Language. Figure 12 illustrates this two-way extensibility of the BOM development. In the static and dynamic BOM, many computed and derived properties have been proposed and implemented for the following purposes: evaluating building circulation and spatial programming. There are many challenging issues in both lateral and vertical extensibility according to the domain and scope of the 'building environment rule and analysis'. Therefore, BOM could have multiple model views with different conventions. In other words, the current implementation and applications described in the following chapter is one of the model views of BOM for evaluating building circulation and spatial program. The goal is to provide easy access to the concepts of a domain, for rule checking.

5 Summary

This paper proposes the BERA Object Model (BOM): a human-centered abstraction of the complex state of real-world building models, rather than the computation-oriented abstraction which is generally intended to cover broad-ranged issues. BOM is one of the key concepts to implement the building environment rule and analysis language. By using BOM, users can enjoy the ease of use and portability to the pre-defined BIM data, rather than sophisticated and platform-dependent ways. A newly proposed BOM data structure has been implemented to cover spatial objects within the scope of this paper and development which focuses on evaluating building circulation and spatial programming. This paper also has described its open-endedness to cover the broader type of building object and its properties. However, the feature is another challenge to define generic and valuable BOM as it grows more detailed. The good example of an open-endedness relevant issue is both lateral extensions such as structural building elements and the vertical extensions and additional properties for existing BOM objects In the BERA Language Tool implementation described in this paper, many computed and derived properties have been proposed and implemented for different purposes, as well as some basic data obtained directly from the given building model. These properties are available to users by dot-notations that are easy to read and write.

In this paper, the BERA Language implementation process has been carefully demonstrated and tested. The ultimate testing and evaluation of the language per se, however, will come from language users, as many users as possible. The open-ended testing, feedback system and support arrangement is planned for updated and upgraded BERA Language and its Tool. The authors expects that more development, especially on top of other types of building modeling platforms, have to be carried out by several entities including the authors so that there are constant contributions to the AEC industry and academia. Expected contributions of the development and use of BERA Language can be summarized as follows:

- The BERA Language will allow for automated building design review and rule checking of BIM models to come into wide use.
- The BERA Language is effective not only for the purpose of design rule checking, but also for various design analysis purposes. In other words, the rules in the BERA Language can be one of many possible user-defined rules even if they are not relevant to existing real world rules. The BERA Language provides a massive analysis method for many building models in an efficient way
- This is the first attempt to develop a BIM domain-specific programming language focusing on building design issues. Therefore, we expect the BERA Language design and implementation to be a model for other domain-specific languages in other domains.
- Within the scope of this study, the initial implementation focuses on spatial objects, group of spaces, circulation paths, their properties, and relations. This implementation will be a basic foundation that can be extended to other various building elements to cover other types of building environment rules and analysis as BERA literally implies

The BERA Language Tool implementation described in this paper is one of the outcomes of building environment rule and analysis – BERA. Development work has just begun: it is open-ended and still growing. The authors believe that the proposed BERA Language and its tool development have a positive and

active influence upon current and future BIM-enabled applications in various disciplines.

References

1. Eastman, C.M.: From blue print to database. Economist, Economist Technology Quarterly June 7, 18–22 (2008)
2. Sanguinetti, P., Abdelmohsen, S., Lee, J., Lee, J.-K., Sheward, H., Eastman, C.M.: General system architecture for BIM: An integrated approach for design and analysis. Adv. Eng. Inform. **26**(2), 317–333 (2012)
3. Lee, J.-K., Lee, J., Jeong, Y.-s., Sheward, H., Sanguinetti, P., Abdelmohsen, S., Eastman, C.M.: Development of space database for automated building design review systems. Autom. Constr. **24**(2012), 203–212 (2012)
4. Zhang, S., Teizer, J., Lee, J.-K., Eastman, C.M., Venugopal, M.: Building information modeling (BIM) and safety: Automatic safety checking of construction models and schedules. Autom. Constr. **29**, 183–195 (2013)
5. Scott, M.L.: Programming Language Pragmatics, 2nd edn. Morgan Kaufmann Publishers (2005). ISBN 978-0126339512
6. ECMA International.: ECMA International. http://www.ecma-international.org/ (2010). Accessed Feb 2010
7. ECMA International.: Standard ECMA-262 - ECMAScript Language Specification. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf (2010). Accessed Feb 2010
8. ISO 16262: ISO/IEC 16262:2002, Information technology - ECMAScript language specification. http://www.iso.org/iso/catalogue_detail.htm?csnumber=33835 (2010). Accessed Feb 2010
9. W3C: World Wide Web Consortium: W3C, an international community where member organizations, a full-time staff, and the public work together to develop Web standards. Led by Web inventor Tim Berners-Lee and CEO Jeffrey Jaffe. http://www.w3.org/ (2010). Accessed Nov 2010
10. W3C: W3C Cascading Style Sheets (CSS). http://www.w3.org/TR/CSS1/ (2010). Accessed Feb 2010
11. Web2.0: Web 2.0 by Paul Graham: Does Web 2.0 mean anything? http://www.paulgraham.com/web20.html (2010). Accessed Nov 2010
12. W3C: Extensible Hyper Text Markup Language 2 (XHTML2) Working Group Home Page, also contains old standards for HTML4, XHTML1.0, etc. http://www.w3.org/MarkUp/ (2010). Accessed Nov 2010
13. XHTML: Extensible Hyper Text Markup Language (XHTML) Reference. http://xhtml.com/en/xhtml/reference/ (2010). Accessed Nov 2010
14. W3C: Extensible Markup Language: XML. http://www.w3.org/XML/ (2010). Accessed Nov 2010
15. W3C: Scripting and AJAX. http://www.w3.org/standards/webdesign/script.html (2010). Accessed Nov 2010
16. JQuery: jQuery Project, a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. http://jquery.org/ (2010). Accessed Nov 2010
17. Netscape Communications: The Netscape Archive. http://browser.netscape.com/ (2010). Accessed Feb 2010
18. W3C: W3C Document Object Model (DOM). http://www.w3.org/DOM/ (2010). Accessed Feb 2010
19. Lethbridge, T.C., Laganiere, R.: Object-oriented Software Engineering. McGraw-Hill (2005). ISBN 978-0073220345
20. Gay, J.: The History of Flash, Macromedia Showcase, Jonathan Gay, 2001. http://www.adobe.com/macromedia/events/john_gay/page02.html (2010). Accessed Feb 2010
21. ActionScript: Adobe Inc., Action Script, a scripting language based on ECMA script for controlling Adobe Flash Media (FLA/SWF files). http://www.actionscript.org/ (2010). Accessed Feb 2010
22. W3C: W3C Cascading Style Sheets (CSS). http://www.w3.org/TR/CSS1/ (2010). Accessed Feb 2010
23. The Open Group: Regular Expressions, The Single UNIX Specification, Version 2, 1997. http://www.opengroup.org/onlinepubs/007908799/xbd/re.html (2010). Accessed Feb 2010
24. Mashey, J.R.: New programming languages are born every day. Why do some succeed and some fail? ACM Queue Volume 2, Issue 9 (December/January 2004-2005) - Languages, Levels, Libraries, and Longevity (2004)
25. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Languages, Pragmatic Bookshelf (2008). ISBN: 978-0978739256
26. Lee, J.-K., Eastman, C.M.: Building Environment Rule and Analysis (BERA) Language, Under Review (2014)
27. Lee, J.-K., Eastman, C.M., Lee, Y.C.: Design Review Applications for the Spatial Program and Building Circulation using BERA Language, Under Review (2014)
28. Scott, M.L.: Programming Language Pragmatics, 2nd edn. Morgan Kaufmann Publishers (2005). ISBN 978-0126339512
29. Parr, T.: Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages, Pragmatic Bookshelf (2009). ISBN: 978-1934356456
30. Parr, T.: ANTLR, Another tool for language recognition, http://www.antlr.org. Accessed Feb 2010
31. Parr, T.: StringTemplate, www.stringtemplate.org. Accessed Feb 2010
32. Parr, T.: Soapbox: Humans should not have to grok XML - Answers to the question When shouldn't you use XML? IBM developerWorks, Terence Parr, (2001), http://www.ibm.com/developerworks/xml/library/x-sbxml.html Accessed Feb 2010
33. Dijkstra, E.W.: On the role of scientific thought. In: Dijkstra, E.W. (ed.) Selected Writings on Computing: A Personal Perspective, pp. 60–66. Springer-Verlag New York, Inc., New York (1982). ISBN 0-387-90652-5
34. MDA.: Model driven architecture (MDA), OMG Architecture Board ORMSC, OMG document number ormsc/2001-07-01 (2001)
35. Kent, S.: Model Driven Engineering, Integrated Formal Methods. Lect. Notes Comput. Sci. **2335/2002**, 286–298 (2002). doi:10.1007/3-540-47884-1_16