

# DEVELOPMENT OF A RULE-BASED SYSTEM FOR AUTOMATED BIM CODE-CHECKING

GIUSEPPE MARTINO DI GIUDA<sup>1</sup>, ANDREA BONOMI BOSEGGA<sup>2</sup> and  
PALEARI FRANCESCO<sup>2</sup>

<sup>1</sup>*Dept of Management, Università degli studi di Torino, Torino, Italy*

<sup>2</sup>*Dept of Architecture, Built Environment and Construction Engineering, Politecnico di  
Milano, Milano, Italy*

This paper aims to describe a method for code-checking integrated into a BIM-based process. Recent developments in the field of model checking, made possible by the increased level of maturity of information modelling, open the possibility to facilitate regulatory controls. However, the automation of this process requires the definition and classification of rules, which represent the translation of regulatory requirements into a computer-based language. This activity, based on Italian legislation, represents the first step to propose a system that aims at integrating BIM models with the rule dataset. For this reason, this paper analyses the different types of queries through rules. These rules must recall principles of generality, replicability, consistency, uniformity. All these requirements are summarized in a structured spreadsheet and compared with the information contained in the BIM model, through a tool implemented within Dynamo software and facilitated by the use of scripts in Python language. The results of this process can be represented in the model and in the spreadsheet for an immediate visualization. This method allows a rapid and detailed control capable of highlighting the potential of information modelling and its integration.

*Keywords:* Building codes, Dynamo, Python scripts, Query types, Requirements, Rule dataset.

## 1 INTRODUCTION

BIM process has reached a level of maturity such that it is increasingly used in the design and construction processes (Solihin 2015). This allows a finer control over the entire process, both for new buildings and for the management of existing ones. Despite that, the checking process of building models to evaluate their conformity to existing regulations is mainly done manually, due to some difficulties that make the automation of this task very challenging (Eastman 2009).

The main issue is the huge amount and variety of existing regulations; each country has its own codes and guidelines for different phases, disciplines and intended use of the building. This extends the time and expertise required to manage all the specifications and requirements of the building codes and make the correct design choices. Nonetheless, the same process has to be done by the authority who has the responsibility of granting the building permission (Bus 2018). A core phase is the iteration through multiple design alternatives to investigate different solutions to problems, while maintaining the compliance with regulatory specifications and finding errors in time. This activity is crucial, and causes some inconsistencies to the overall design due to human errors.

An automated system of code-checking can facilitate and speed-up this work, while incrementing the coherence and reliability of the entire project.

Several software solutions try to avoid this problem, but most of them are based on a black-box approach, hiding the internal logic and algorithms to the end user and hindering changes (Preidel 2018). This paper aims to find a different path, and define a general method of code-checking based on the principles of replicability, coherence, uniformity and automation, focusing on the management of BIM-based models.

## 2 METHOD

### 2.1 General Approach

As investigated by Eastman (2009), a rule-based system should support some classes of functionalities like rule interpretation, building model preparation, rule execution and report of checking results. Each one serves a specific task, and should be investigated separately. The authors stated that this conceptual separation in classes serves their purpose of surveying the checking systems based on rules, but they expect new issues to emerge.

Following studies try to further define the challenges of automated code compliance checking; Aimi (2017) highlights the importance of building codes interpretation as “the most vital and complex stage”. So, a fundamental part is to parse the implicit and contradictory statements and make them explicit (Bus 2018). Assuming that, the adopted structure subdivides the problem into four main tasks, as suggested by Eastman (2009) and Bus (2018):

- The definition of the working environment;
- The definition of the code-checking requirements, to consistently prepare the BIM models and the contained data;
- The interpretation and translation of building codes into non-ambiguous rules for the construction of a structured collection;
- The report of the check's results.

### 2.2 Working Environment

#### 2.2.1 *Object-oriented approach*

There are numerous ways to develop the proposed system, and depending on the initial choices its implementation success can drastically change. As a main principle, separating the concerns keeps the system cleaner, and ensures its repeatability (Hjelseth 2016). For this reason, two main elements are identified: data sources and the tool that executes the check. This separation ensures the autonomy of information whose degree of complexity and detail can widely change according to project requirements, guidelines and building codes. On the other hand, it allows to choose the software that best fits users' needs, both for the realization of the BIM model and for the implementation of the code-checking system.

Moreover, two main data sources can be detected:

- The BIM model, containing all the geometries and information defined by project requirements;
- The rulesets, a structured collection of all the rules extracted by the regulations.

This approach recalls the structure of an object-oriented system (Yang 2001), based on the subdivision of attributes in classes. These classes are represented by the various legislative areas, such as structures, interiors and exteriors spaces, installations or energy consumption aspects. In addition, each class can be easily extended to include other regulations or custom guidelines.

### **2.2.2 Principle of the white-box**

As previously highlighted, existing commercial solutions lack both transparency and customization in analyzing input data due to their black-box approach. Although they are very reliable, the use of these systems entails that professionals have to hand over their responsibilities to hidden algorithms, generating a lack of trust over the results. In addition, they can lead to hidden errors caused by misuse of the tool, or inoperable software because of the lack of some building codes. These issues can be overcome by adopting a white-box approach and by making the whole system accessible. Users can freely edit any part of the system, fix errors, update rules and customize it according to their specific needs, resulting in a trustworthy and complete approach.

### **2.3 Model Requirements**

Albeit often neglected, the definition of starting requirements can be crucial for the success or failure of the checking process. If the model contains less data than the amount required to successfully run the check, the latter will be incomplete and easily led to failure. Also, the overload of information is a useless waste of time, while only a part of it will be queried during the regulatory controls. Lastly, if the model is incorrectly prepared, the checking process will struggle to find the correct slots where the data is located, resulting in a check failure. So, the purposes of the model and the checks required have to be defined at the beginning of the process, along with the project requirements, to avoid malfunctioning and time loss. Each object modeled should include only the exact amount of information required, other than its geometry (Eastman 2009). For example, building codes often require checking the localization of an object: in that case, the object can be modeled as a box.

### **2.4 Regulatory Translation and Ruleset**

Translating building codes into formal rules is the core part of any code-checking tool.

Previous studies (Eastman 2009) show that the most common method to translate regulatory text into rules is the first order predicate logic. Each rule is a predicate that can be evaluated as true or false. All prescriptive regulations generally require this type of checks, but only measurable prescriptions can be abstracted to the level of rule; this issue can be a weakness, but it can also help to simplify regulatory texts to their core aspects.

#### **2.4.1 Rule's structure**

A rule is the building block of this system; it is an autonomous entity and does not rely on other rules (Figure 1). Instead of being related to a specific environment, it can be implemented into any existing software, as it supports both simple and complex logic statements and it supports the model specifications used by the software.

There are mainly two ways to define a rule: with parametric tables or using specific computer programming language (Eastman 2009). Both methods have pros and cons, such as the straightforwardness of the first one and the potential that derive from the latter. On the other hand, tables can't represent big ranges of conditions, and learning a programming language can be a major obstacle to non-programmers. However, these two methods can be used together, as coding competences are being included in school curricula in recent years.

The process of defining a rule should follow some precise steps. The first one concerns the extraction of a semi-formal rule from the regulatory plain text. This one, proposed by Bus (2018), serves the purpose of keeping it still readable to humans; also, it allows to maintain a high

level of abstraction, independent from the specific implementation. At this level, each parameter has to be submitted to “if-else conditions”. This structure also fits into a table: each row contains a parameter, and columns are used to represent the condition that controls parameter’s values to be respected. For example, columns can represent architectural typologies with different threshold values, or distinguish between different building codes and guidelines.

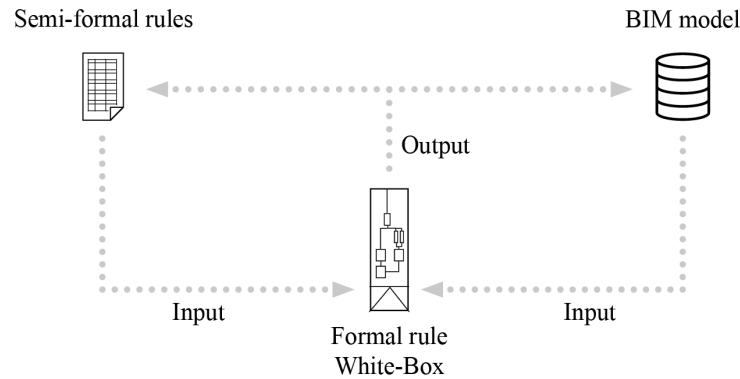


Figure 1. Rule structure.

Then, semi-formal rules can be converted into formal rules based on the working environment defined. As a general statement, formal rules are the core elements. They search for specific data packages from both the model and the semi-formal rule table, they compare the retrieved information by using predefined conditions derived from the formal-rules and they return the results as a “passed” or “not passed” check. Depending on the software, it can be implemented by using visual programming or scripting languages.

#### 2.4.2 Rule classes and patterns

The process of translating plain text into rules highlights the existence of some patterns. Based on Solihin (2015), three classes are identified as shown in Figure 2.

The first class collects rules based on explicit data. It is the most straightforward one, as it directly checks the information contained into the object’s parameters. Information can be on any data type, such as numeric values, Boolean values or textual strings. The latter can be properly managed by defining in advance the exact text string to use. These solutions guarantee unambiguity of strings and avoid checking errors caused by inconsistencies.

The second class groups rules based on derived data. It is the more complex class to implement, as it strongly relies on the data structure of the software adopted. These rules derive the information to be checked from the combination of multiple parameters. They use custom algorithms to process the data obtained from specific parameters and derive the information requested. Examples of these rules are the sum of areas, distances or average values.

The third class extends the first two and collects the rules based on external data inputs. For this reason, both explicit and derived data rules may fall in this class if they need external inputs to process the BIM model information. For example, to find the object located in a specific room the system has to recognize the correct room object. This can be done by introducing an encoding parameter into the model. However, since encoding systems are project-specific, they should also be included into the rule as external inputs.

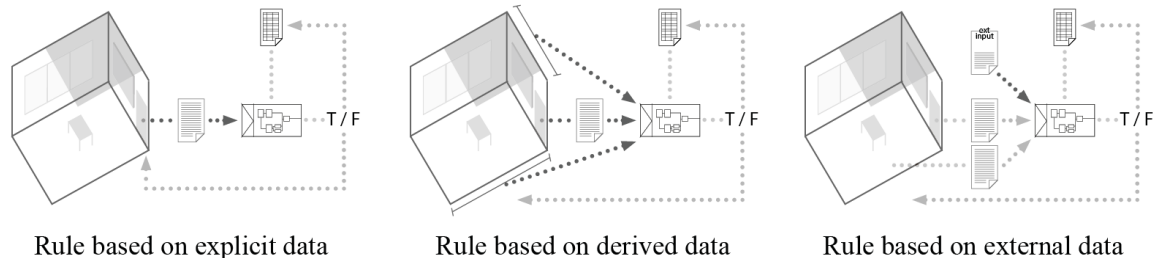


Figure 2. Classes of rules.

### 2.5 Check Reports

As a last step, the results of the rules execution have to be shown to the user. A coherent approach is to return the output data to the same location of the input one, to create a direct relation between the two types of information. Therefore a rule must write the result of the check both into the BIM model, to provide designers with important information, and into the semi-formal rule's table.

### 3 RESULTS

This system has been tested on a case study, concerning the renovation of an Italian primary school. For this reason, Italian's school regulations have been considered, such as fire safety, accessibility, and energy performances legislative requirements. For each category, plain text has been synthesized, interpreted and translated in semi-formal rules and for a better understanding of the presented system, an example of rule is introduced. The software adopted are the Autodesk Revit package along with the integrated open source visual programming tool of Dynamo, further extended with custom nodes written in Python by using the Revit Python API (Figure 3).

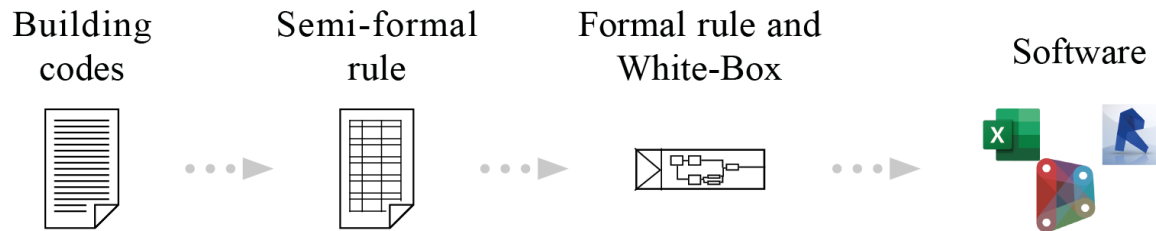
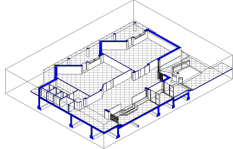
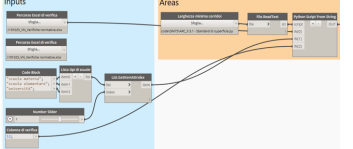


Figure 3. Implementation process.

According to the regulations, minimum surface areas are defined for all the spaces. These can vary depending both on school grade and on number of classes. The semi-formal rule's table shows the school grade on the columns, and with a simple function it calculates all minimum areas by providing the number of classes as a required input value. This way, the rule implemented in Dynamo and Python must require two input values to allow the execution: the school grade and the code of the room. Then, it can search for the correct value by iterating through the model data, executing the check and returning the result (Table 1).

Table 1. Implementation example.

1. BIM model (Autodesk Revit)	2. Semi-formal rule (Excel table) - columns classify params by school grade.	3. Formal rule (Dynamo Revit) - both visual and Python scripting.																																	
	<table border="1"> <thead> <tr> <th></th> <th>Scuole di ogni grado di tipo 2</th> <th>Scuole di ogni grado di tipo 3</th> </tr> </thead> <tbody> <tr> <td colspan="3"><b>SCALE, ASCENSORI E MONTACARICHI</b></td> </tr> <tr> <td colspan="3"><b>SCALE</b></td> </tr> <tr> <td>Larghezza minima [cm]</td> <td>120</td> <td>120</td> </tr> <tr> <td>Ramppe rettilinee</td> <td>51</td> <td>51</td> </tr> <tr> <td>Inclinazione</td> <td>60</td> <td>60</td> </tr> <tr> <td>N. minimo gradini</td> <td>3</td> <td>3</td> </tr> <tr> <td>N. massimo gradini</td> <td>15</td> <td>15</td> </tr> <tr> <td>Piatta gradini</td> <td>rettangolare</td> <td>rettangolare</td> </tr> <tr> <td>Altezza acataa max [cm]</td> <td>17</td> <td>17</td> </tr> <tr> <td>Profondità pedana min [cm]</td> <td>30</td> <td>30</td> </tr> </tbody> </table>		Scuole di ogni grado di tipo 2	Scuole di ogni grado di tipo 3	<b>SCALE, ASCENSORI E MONTACARICHI</b>			<b>SCALE</b>			Larghezza minima [cm]	120	120	Ramppe rettilinee	51	51	Inclinazione	60	60	N. minimo gradini	3	3	N. massimo gradini	15	15	Piatta gradini	rettangolare	rettangolare	Altezza acataa max [cm]	17	17	Profondità pedana min [cm]	30	30	
	Scuole di ogni grado di tipo 2	Scuole di ogni grado di tipo 3																																	
<b>SCALE, ASCENSORI E MONTACARICHI</b>																																			
<b>SCALE</b>																																			
Larghezza minima [cm]	120	120																																	
Ramppe rettilinee	51	51																																	
Inclinazione	60	60																																	
N. minimo gradini	3	3																																	
N. massimo gradini	15	15																																	
Piatta gradini	rettangolare	rettangolare																																	
Altezza acataa max [cm]	17	17																																	
Profondità pedana min [cm]	30	30																																	

## 4 CONCLUSIONS

This approach summarizes practical code-checking related issues, defining a general and replicable method, tested into a specific context. Its applicability can be further extended. The system has a great potential over the checking of prescriptive codes, due to its implementation straightforwardness, that does not need any special implementation other than a parameter lookup from both the model and the semi-formal rule’s table. Nonetheless, thanks to its modularity the system allows a deeper implementation of rules, not only for prescriptive parameters but also for qualitative and quantitative ones. The white-box approach allows the creation of a database of rules and semi-formal tables, and represents the basis for a growing collection of working rules. In addition, the classes defined are general constraints to define more complex rules: this paper does not directly take into account performance-based regulations, but they can generally be translated into rules based on derived attributes or on external data. The process can require more time, but it leads back to some basic rules shaped by more complex conditions. The main limitations are the knowledge of the software used, their API (Application Programming Interfaces) and the level of integration between the BIM model and the implemented rules. Overcoming these difficulties can lead to technically advanced rules, such as geometry and context-based analysis or path-finding tasks, and can facilitate the checking task.

## References

- Aimi, S. I., Kherun, N. A., and Noorminshah, A. I., *A Review on BIM-Based Automated Code Compliance Checking System*, 5th International Conference on Research and Innovation in Information Systems (ICRIIS), Langkawi, doi: 10.1109/ICRIIS.2017.8002486, July 16-17, 2017.
- Bus, N., Roxin, A., Picinbono, G., and Fahad, M., *Towards French Smart Building Code: Compliance Checking Based on Semantic Rules*, LDAC2018 6th Linked Data in Architecture and Construction Workshop, London, United Kingdom, June 19-21, 2018.
- Eastman, C., Lee, J. M., Jeong, Y.-S., and Lee, J.-K., *Automatic Rule-Based Checking of Building Designs*, Automation in Construction, Science Direct, 18, 1011–1033, December, 2009.
- Hjelseth, E., *Classification of BIM-Based Model Checking Concepts*, Journal of Information Technology in Construction (ITcon), Special issue: SIB W78 2015 Special track on Compliance Checking, ITcon, 21, 354-369, November, 2016.
- Preidel, C., and Borrmann A., *BIM-Based Code Compliance Checking*, Building Information Modeling, Springer, 367-381, September, 2018.
- Solihin, W., and Eastman, C., *Classification of Rules for Automated BIM Rule Checking Development*, Automation in Construction, Elsevier, 53, 69–82, May, 2015.
- Yang, Q., and Li, X., *Representation and Execution of Building Codes for Automated Code Checking*, Proceedings of CAAD Futures, Eindhoven, Netherlands, doi: 10.1007/978-94-010-0868-6\_24, July 315–329, 2001.