Refinement of the Visual Code Checking Language for an Automated Checking of Building Information Models Regarding Applicable Regulations

Cornelius Preidel and André Borrmann

Chair of Computational Modeling and Simulation; TUM Department of Civil, Geo and Environmental Engineering, Technical University of Munich, Arcisstraße 21, 80333 Munich; Email: cornelius.preidel@tum.de

ABSTRACT

With Building Information Modeling becoming mature, this technology provides several new possibilities for improving the checking processes of the design planning regarding the applicable codes in the construction industry. Code compliance checks are essential since they standardize the functions a building must fulfill finally. Currently, these controls are cumbersome and error-prone and performed to a large extent manual by the planning consultants as well as authority officers. To enable an automation, the contents of the applicable guidelines have to be translated into a digital format, which can be read, interpreted and applied by machines. Various approaches focused on a textual or hard-coded translation of the guidelines. To increase the involvement of non-programming users, we introduced the Visual Code Checking Language, which uses a visual notation to represent the content of directives machine- as well as human-readable. In the presented paper, the authors present a refinement of the VCCL to increase the coverage rate of the translation capabilities.

INTRODUCTION

Nowadays, the compliance of building designs with applicable requirements and regulations is checked largely manually. As a rule, a large number of international, national and regulations must be taken into account by the planning consultants as well as building authority officers to be sure that the building fulfills all functions properly (Boken and Callaghan 2009; Eastman et al. 2009). Therefore, the checking and review cycles can take a long time and thus contribute to the increase in the cost of the entire process. Today this process has a very low degree of automation, and so the process is laborious, cognitively challenging and error-prone for the users. To overcome this issue, the authors introduced the Visual Code Checking Language (VCCL) to provide users an instrument which they can use to translate the contents of codes visually and thus formalize them. This digital neutral representation can be used to partially or even fully automate the checking process, which is also referred to as *Automated Code Compliance Checking*. The main advantage of this approach is that the readability of the encoded checking process is preserved for the users at the same time. Non-transparent black-box procedures, such as hard-coded or very complex approaches, which require programming knowledge, cannot be translated or adapted by the user. Since the user can not verify or test the results, his acceptance is dramatically reduced (Preidel and Borrmann 2015).

The authors have formulated basic requirements, which must be fulfilled by an approach for an Automated Code Compliance Checking: The method should be (1) capable of the expression of geometric as well as non-geometric constraints and requirements, (2) accessible for domain experts

with limited software development knowledge, and (3) independent from the employment of a particular code checking engine (Preidel and Borrmann 2016).

STATE OF THE ART

In recent years, the number of efforts for an *Automated Code Compliance Checking* has increased. A basic overview of state of the art in this field including the most important open research questions is given by Eastman et al. (2009). The authors report a significant need for studies in this application area and focus especially on the separation of the representation of rules as well as the techniques required for processing and checking them.

Several approaches are focusing on rules as procedural code embedded within a code checking system (Ding et al. 2006; Eastman 2009; Eastman et al. 2009; Solihin 2004). In this way, vast areas of the codes can be translated and formalized directly, but the code is not accessible for third-party developers or domain experts anymore so that its correctness is thus not verifiable. These approaches are also referred to as "black box" implementations, whose acceptance is usually not very high among the domain experts (Nisbet et al. 2008). An example of such a black-box implementation is the system CORENET e-PlanCheck, which is a platform hosted by the Singaporean government for checking the compliance of digital building models with design requirements in the areas of building control, barrier-free access and fire safety (Solihin 2004). The methods of this system are summarized in the FORNAX library as black-box implementations and have been developed and maintained by a private company.

Another major commercial solution is the Solibri Model Checker (SMC), which is a Java-Application for BIM-based model quality and code checking. The SMC was published by the Finnish company Solibri in 2000. All processes within this application are based on information of IFC-Models, which are mapped onto an internal data model. The core of the SMC's checking routines is the Ruleset Manager, which provides a basic library of 42 single rule templates. Such a rule template represents a hard-coded standard checking procedure, which can be adjusted by a limited number of given parameters. These rule templates can be composed or adjusted by the user regarding his individual requirements. Also, these rules cannot be represented using a human-readable external format but have to be implemented using a native data format.

Another genre of research approaches is focusing on the development of a Natural Language Processing (NLP) system for an automated extraction of the rule knowledge from flow texts of guidelines such as in (Zhang and El-Gohary 2016a), (Zhang and El-Gohary 2016b) or (Uhm et al. 2015). However, these approaches are currently limited, since the representation of complex knowledge, such as geometric constraints or operators, is not possible yet. Furthermore, these systems are designed for full automation and can only be checked, managed and adapted by specialist experts with programming knowledge.

Furthermore, various approaches are based on the development of a domain-specific textual query language. As an example, Lee (2011) developed the Building Environment and Analysis Language (BERA). In (Lee et al. 2015) this language is specially applied for the encoding of more complex rules regarding evaluating a building's circulation and spatial programs. BERA provides a set of spatial operators for the definition of standards in the context of these application areas. However, the developed language lacks the desired generality to cover sufficiently any content, but as a first essential language-based approach it forms a highly significant foundation and a point of departure.

A promising different approach by (Solihin and Eastman 2016) introduces a visual translation method based on conceptual graphs representing design rules. The method is characterized by the fact that it tries to find a balance between readability for the user and a clear and stringent interpretation and translation of the rules.
In summary, it can be stated that significant research has been conducted regarding the *Automated Code Compliance Checking* so far. However, several approaches introduce methods, which base on hard-coded methods, which results in a severe lack of transparency and flexibility of the encoded rule system. Only a few approaches take into account the involvement of the user in the process, such as an employment of a computer-processable intermediate representation of the building code.

METHODOLOGY

As described in the previous sections, the challenges of an Automated Code Compliance Checking have not yet been solved holistically. Therefore, the authors have introduced a novel approach based on Visual Programming in (Preidel and Borrmann 2015). Following, the methodology will be explained briefly, followed by the major improvements in the language.

Visual Programming. In recent years, Visual Programming Languages (VPL) have made inroads into the field of digital construction. Users can employ these languages as a tool to make repetitive work or the creation of variants and their evaluation a lot easier without the need for detailed programming knowledge. A visual language is defined as a formal language with visual syntax and semantics (Schiffer 1998). It describes a system of signs and rules on the syntactic and semantic level with the help of visual elements, which are more readily understandable for non-professional programmers. Visual programming languages are often referred to as flow-based since they represent complex structures as a flow of information (Shu 1988). An essential distinguishing characteristic of different visual programming languages is the granularity. This granularity describes how finely the specific functionalities are resolved, i.e. whether functions within a node are entirely encapsulated or whether each sub-step is available and visible as a separate node. Encapsulation (low granularity) reduces the number of elements present in the workspace and contributes to the clarity, handling, and comprehensibility of the overall system. At the other end of the scale (fine granularity), the resulting canvas is a more detailed representation of the information process in which the user can access and adapt each step as required.

Visual Code Checking Language (VCCL). The VCCL is a domain-specific programming language designed for the formulation of checking and verification routines, which are contained in standards or guidelines. Based on digital building information, the VCCL is intended to perform compliance checks automated or semi-automated, which increases the efficiency and quality of the overall process significantly. The scope of an application implies special requirements for the design of the VCCL, which makes this language different from query languages. As with the query languages, the VCCL also needs to extract information from the building model, but these must be prepared dependent on the relevant code, more extensively. Furthermore, rule knowledge has to be represented, e.g. which requirement must be compulsory, under certain circumstances or not be fulfilled. For this reason, several operators and methods were introduced for the VCCL, which distinguishes the visual language from other representatives. A first version of the VCCL was presented in (Preidel and Borrmann 2015).

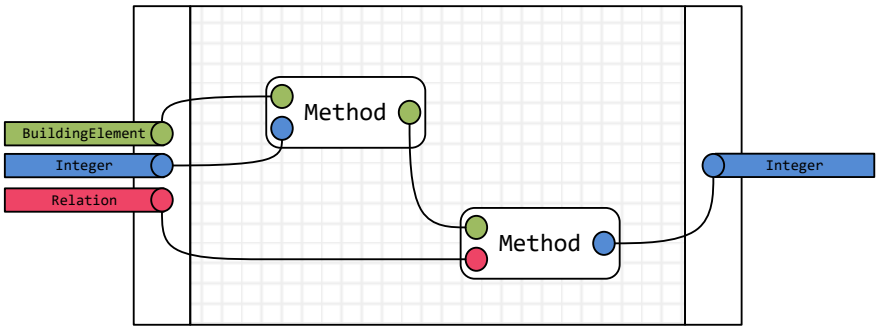

Figure 1. Generic VCCL graph example

Lessons Learned - Refinement of the VCCL. The authors have recognized the necessity that the language has to be adapted for several reasons: In the prior version of the VCCL, objects were displayed separately, and therefore graphs became overly complex. Furthermore, the design of the language was not able to represent complex control structures or nested graphs, which are necessary for the representation of challenging guidelines. Therefore, the design of the language was adopted and shall be presented in the following: The VCCL is modular and strong-typed, among other things. For the description of information processes, the VCCL provides basic elements: (1) Methods (rectangle nodes) representing well-defined operations, (2) Ports (Input- and Output Circles for the Methods) representing the incoming and outgoing data type, and (3) Directed Edges, which connect the methods. The user can compose these elements on a canvas to a VCCL graph according to their requirements and thus represent the desired checking process. For the creation of a VCCL graph, the user is provided with a three-part control, which is divided into an input and output as well as a process environment. To define a distinct starting and ending point of each VCCL program, the global input and output area have some user-defined ports, which serve as an initial information source and resulting ending point for the defined routine. When creating a VCCL program, the user has to define which type of information is provided as the initial input and which kind of information is produced as the final result of the program. For a more detailed introduction and discussion of the VCCL, the reader is referred to (Preidel and Borrmann 2016). A generic example of a VCCL graph is shown in Figure 1.

Granularity and Modularization. The VCCL is intended to make any process step the user might be interested in visible so that he can have an insight on demand. Therefore, the VCCL follows the principle of realizing the finest processing granularity possible. To avoid overly broad and complex VCCL networks, we introduce a nesting approach which makes use of the global input and output areas. Within the VCCL environment, a method describes an operation, which can also be displayed as a separate VCCL graph and therefore is shown as a nested control. In textual programming languages, this concept is implemented with functions and subroutines. A nested VCCL program processes some defined operation steps which are hidden on the upper level to reduce the visual complexity of the overall VCCL program. By defining input and output ports for the nested program, it can be saved as single graph instance and re-used in any given VCCL program. In this way, a hierarchic VCCL library is built-up. A generic example for the modularization principle is shown in Figure 2.

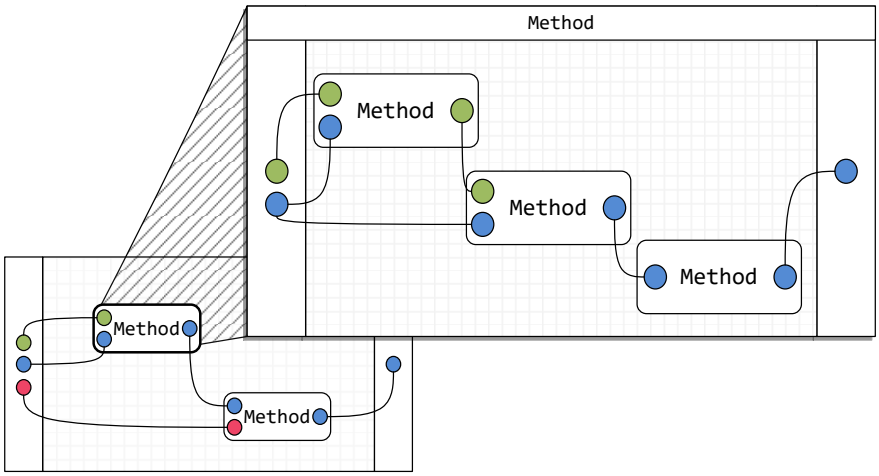

Figure 2. Granularity Example

As a prerequisite, basic methods must be given for user, so that he can compose the hierarchic structure. These given methods describe basic fundamental operations, whose semantics are unambiguously well defined. Therefore, these methods are called atomic methods. These include, among others mathematical operators, geometric operators, relational algebra operators, or access operators for the extraction of information of building models. These methods represent a certain "black-box level", from which we assume that it is acceptable for a given application area, where deeper control and insight is neither desired nor necessary, and too much effort would be created for implementing the provided functionality using VCCL instead of a standard programming language. An example, an acceptable black-box atomic method is the evaluation of geometric information such as the computation of the shortest distance route for a given floor plan. It may be assumed that a user is not interested having insight in such processes. However, it is important that atomic methods be characterized by genericity so that the methods can be used for different purposes. For sure, various other black box limitations have to be considered for various applications and varying levels of experience. Another major advantage of the modularization is a representation of control flows, which allow describing more complex systems. Most of the existing imperative programming languages provide at least the most rudimentary elements of control flow such as conditional branching or iterations. Conditional branching is typically realized in textual programming languages using if statements while iterations are implemented using loops. The VCCL implements these control structures as derivations of nested methods. In this way, each VCCL program can be tested for validity separately. For details, the reader is referred to (Preidel and Borrmann 2016).

Formalization. Encoded VCCL graphs do not necessarily have to be visualized, but can also be stored in XML-based format. So far, this format has largely been independent and has not been harmonized with other, standardized formats. Since this feature is essential for the exchange between different project stakeholders, it is planned to streamline the developments with the current standardization in this application area. Although there are already standards for the description of guideline knowledge such as LegalRuleML or LegalDocML, one has not yet agreed on any of these. However, buildingSMART (2016) recently founded the working group Regulatory Room, which aims to find an open format that can represent this knowledge. Although it will presumably take some time until first results will be available, it is planned to align VCCL

developments with the outcomes of this working group and make the VCCL content independently accessible.

APPLICATION EXAMPLE

To proof the increased translation capability and coverage rate of the VCCL, it was applied to the translation of Article 34 Clause 1 of the Korean Building Act (Korea Legislation Research Institute 2008), which is shown in Figure 3.

> *"On each floor of a building, direct stairs leading to the shelter floor or the ground other than the shelter floor shall be installed in the way that the walking distance from each part of the living room to the stairs is not more than 30 meters: Provided, that in cases of a building of which main structural part is made of a fireproof structure or non-combustible materials, the walking distance of not more than 50 meters may be established, and in cases of a factory prescribed by Ordinance of the Ministry of Land, Infrastructure and Transport, which is equipped with automatic fire extinguishers, such as sprinklers, in an automated production facility, the walking distance of not more than 75 meters may be established."*

Figure 3. Content of Article 34 Clause 1 of the Korean Building Act (Korea Legislation Research Institute, 2008)

The modularization approach allows dividing the quite complex regulation into different parts, which reduces the complexity of the visual translation later on. In a first VCCL module program, all exit staircases must be identified for each floor. Therefore, the staircases (rooms, which have stairs inside) have to be checked, if they can be used as exits (rooms, which have exit doors). Afterward, each single floor can be verified for a connection with an identified exit staircase. The encoded VCCL module graph is shown in Figure 4.
In the second step, the computation of the walking distance represents an atomic method how has been described in the methodology section of this paper. We assume that this computation may be represented as a given method, which the user most likely has no interest to have an insight. This approach calculates the maximum walking distance from the floor plan and a respective target point. In Figure 5 the final composed VCCL translation of the guideline is shown. In this graph, the presented VCCL method for the identification of the stairways as well as the atomic walking distance method is used on a higher hierarchical level of the VCCL. At this point, we assume, that the indication whether the main structure of the building is classified as fireproof or is equipped

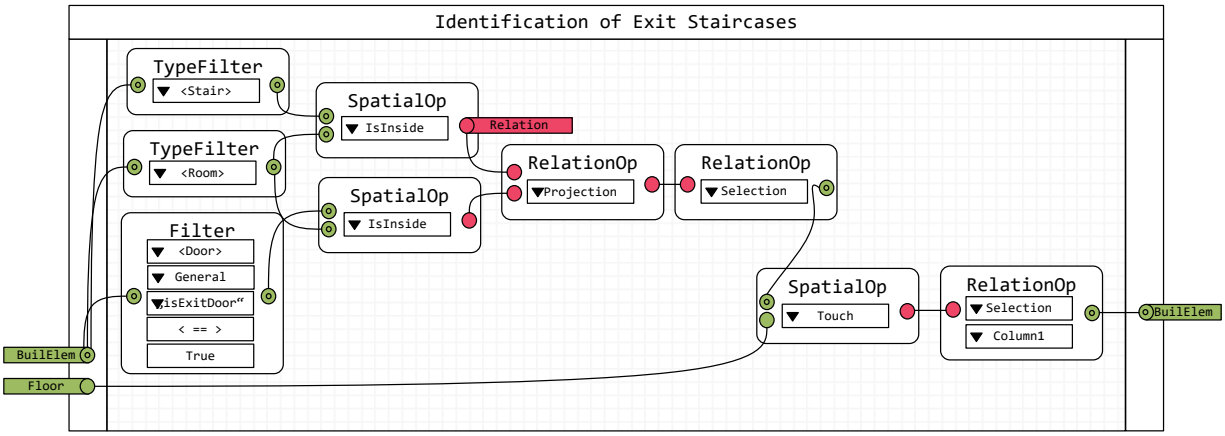

Figure 4. Identification of exit staircases for a selected floor with the VCCL

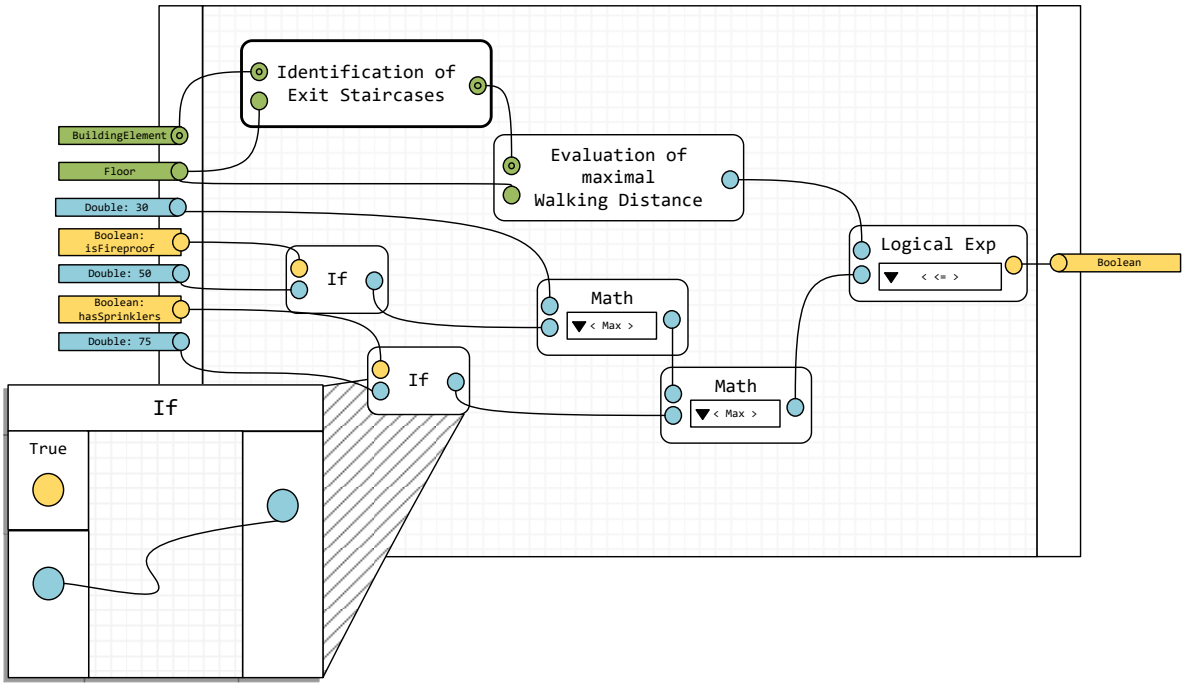

Figure 5. Encoded VCCL representation of Article 34 Clause 1 of the Korean Building Act

with automatic sprinklers is information, which is modeled in the building information model and therefore is given. This information can be obtained either as a user input or evaluated in a higher VCCL program from the information of the underlying building model. In the present case, the affected values are introduced as input variables for the VCCL program. For the translation of the relevant value for the maximum walking distance, the program uses *If* elements. The significant maximum value can be determined using simple mathematical methods.

CONCLUSION AND OUTLOOK

The automated checking of digital building models regarding the applicable building regulations is a major challenge in the construction industry since it is an essential process step which could lead to a significant increase in efficiency. With the introduction of the VCCL, the authors intend to provide a suitable solution that meets the special and individual requirements for this automation. In the first version of the language, the authors noted that changes had to be made to increase the reach and cover rate of this approach. Therefore, this paper presents an enhancement of the VCCL, which describes the modularization of complete or partial VCCL graphs. With the help of this method, the graphic system can be simplified and thus the visual complexity reduced. Since it is possible to translate and present more complex requirements and systems in this way, it can be expected that significantly higher cover rates will be achieved in the future. Further applicability tests in various application areas are planned as a next step.

Acknowledgements. The authors gratefully acknowledge the support of the Nemetschek Group as well as ALLPLAN GmbH for the research project presented in this paper.

REFERENCES

Boken, P., and Callaghan, G. (2009). "Confronting the Challenges of Manual Journal Entries." *Protiviti*, 1–4.

BuildingSmart. (2016). "Regulatory Room." <http://buildingsmart.org/standards/standards-organization/rooms/regulatory-room/> (Aug. 25, 2016).

Ding, L., Drogemuller, R., Rosenman, M., and Marchant, D. (2006). "Automating code checking for building designs - DesignCheck." *Cooperative Research Centre (CRC) for Construction Innovation*, 1–16.

Eastman, C. (2009). "Automated Assessment of Early Concept Designs." *Archit Design*, 10.1002/ad.851, 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*, 10.1016/j.autcon.2009.07.002, 1011–1033.

Korea Legislation Research Institute. (2008). *Article 34 Clause 1*.

Lee, J. K. (2011). *Building Environment Rule and Analysis (BERA) Language,* 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." *J Intell Robot Syst*, 10.1007/s10846-014-0117-7, 507–522.

Nisbet, N., Wix, J., and Conover, D. (2008). "The Future of Virtual Construction and Regulation Checking." *Virtual futures for design, construction & procurement,* P. S. Brandon, and T. Kocatürk, eds., Blackwell Pub, Oxford, Malden, MA, 241–250.

Preidel, C., and Borrmann, A. (2015). "Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling.", International Symposium on Automation and Robotics in Construction and Mining; ISARC, Red Hook, NY.

Preidel, C., and Borrmann, A. (2016). "Towards code compliance checking on the basis of a visual programming language." *Journal of Information Technology in Construction: Special Issue CIB W78 2015 Special track on Compliance Checking*, 402–421.

Schiffer, S. (1998). *Visuelle Programmierung: Grundlagen und Einsatzmöglichkeiten*, Addison-Wesley, Bonn.

Shu, N. C. (1988). *Visual programming*, Van Nostrand Reinhold, New York.

Solihin, W. (2004). "Lessons learned from experience of code-checking implementation in Singapore.", *BuildingSMART Conference* <https://www.researchgate.net/publication/280599027_Lessons_learned_from_experience_of_code-checking_implementation_in_Singapore> (Oct. 6, 2016).

Solihin, W., and Eastman, C. (2016). "A knowledge representation approach in BIM rule requirement analysis using the conceptual graph." *Journal of Information Technology in Construction: Special Issue CIB W78 2015 Special track on Compliance Checking*, 370–401.

Uhm, M., Lee, G., Park, Y., Kim, S., Jung, J., and Lee, J.-K. (2015). "Requirements for computational rule checking of requests for proposals (RFPs) for building designs in South Korea." *Advanced Engineering Informatics*, 10.1016/j.aei.2015.05.006, 602–615.

Zhang, J., and El-Gohary, N. M. (2016a). "Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking." *Automation in Construction*, 10.1016/j.autcon.2016.08.027.

Zhang, J., and El-Gohary, N. M. (2016b). "Semantic-Based Logic Representation and Reasoning for Automated Regulatory Compliance Checking." *J. Comput. Civ. Eng.*, 10.1061/(ASCE)CP.1943-5487.0000583, 4016037.