Towards Rule-Based Model Checking of Building Information Models

C. Sydora[a] and E. Stroulia[a]

[a] Department of Computing Science, University of Alberta, Canada
E-mail: csydora@ualberta.ca, stroulia@ualberta.ca

Abstract –

Designing a building, so that it adheres to all the relevant applicable constraints imposed by construction codes to cultural preferences to the owners' styles and aesthetics, can be a daunting task, requiring many laborious hours of review and modification. Given the increasing adoption of Building Information Modeling (BIM) in the design process, automated model checking is a pragmatic approach to expeditiously identifying errors that may otherwise cause issues later in the building phase. A variety of methods have been proposed, but they are opaque regarding the rules they consider, and they do not allow users to edit these rules. In this paper, we describe a simple, yet extendible, language for specifying building rules and a method for evaluating these rules in the context of a BIM instance, in order to assess the compliance of the building with these rules.

Keywords –

Building Information Modeling (BIM); Design Constraints; Rule Checking

1 Introduction

Building Information Modeling (BIM) has become a key part of the design and management process for Architecture, Engineering, and Construction (AEC). It supports the 3D visualization and design of buildings, while also making explicit the non-geometric properties of, and relationships between, objects. While this is extremely useful and valuable information for the domain experts, it can be an overwhelming amount of data for bigger buildings with complex models. Each additional object in the model implies multiple new relationships between this new object and existing objects, which increase exponentially as the final model takes form. The process of designing buildings can be a complex and error-prone task, given the attention that needs to be paid to each design consideration. Mistakes can be made if a designer is not aware of the ramifications of their design choices and, unless identified early, they can lead to inefficiencies and potentially excessive additional costs in the future [1].

Checking a BIM model against a set of design rules has been a major topic in BIM research for over a decade, and yet no broadly available solutions exist to support rules from a variety of sources, such as governing agencies, handbooks, and builders. While some model-checking software systems exist, they either require that their users possess a strong software-programming knowledge to configure them with rules of interest, or they are black boxes, not configurable at all. Since it is unlikely that all stakeholders will ever be able to agree on a single immutable set of rules, applicable to all buildings, these products are fundamentally limiting the wider adoption of automated model-checking of buildings. Other attempts at automated model checking have taken the Natural Language Processing (NLP) approach, aiming at automatically transforming rules from human-readable specifications into programmatic executable code. While these methods have many benefits in terms of ease of use, there is usually far too much leniency in the written language, which makes it impossible to process automatically and accurately; as a result, these methods are fundamentally limited in their capacity to capture the requirements around compliance checking.

Our methodology is grounded in the intuition that there can be no effective "one-size-fits-all" approach to the problem [2]. In fact, we suggest that model checking be organized around different levels, appropriate for the level of programming expertise of the stakeholder checking the model. Moreover, all levels of experience should be able to work on a single open platform and use the Application Programmable Interface (API) that they feel most comfortable with. This is because there is a trade-off between the complexity of the rules, the expressiveness of the language used to specify them, and the ease of use. While some rules may require intricate functions that can be difficult to formulate, a substantial portion of rules can be described using simple logic and standard geometric relations.

In this paper, we describe a simple, easy to use, logic-based language for describing rules. Our language

is expressive enough to cover a wide array of rules, and we argue that, in principle, it can be extended to broaden its coverage.

The rest of this paper is organized as follows. We first outline some of the previous approaches in the related work in Section 2. Our methodology is described in Section 3. Section 4 contains discussion points on our methodological assumptions and how the language and rules fit in the larger picture. Concluding remarks are in Section 5.

2 Related Work

Solibri. When researching compliance-checking tools, Solibri Model Checker (SMC) [3] is frequently as one of the few tools specifically built for the purpose of checking BIM models. SMC takes as input a building model in form of the BIM industry standard of Industry Foundation Classes (IFC) [4]. While the available rulesets, initially from the Norwegian Statsbygg handbook [5], can be modified by the end user (by combining rule sets and deleting rules), there is limited support for editing rules in the form of changing the parameters (but not the form) of the provided rules. Additionally, there are a few rule templates that can be manipulated, however, full customization of rules can only be done through the SMC API, which is not open.

There are a number of research papers that report how different checks might be implemented using the available rule templates [6], [7], and [8]; however, these are black-box approaches and it is impossible to comment on their accuracy, efficiency, generality and expressiveness.

Country-Specific Implementations. Model-checking tools have been implemented for the purpose of evaluating requirements of governing bodies, with differing levels of success. Singapore's CORENET ePlanCheck has been noted as the most successful implementation, since, at one point, it was mandatory as part of the government's building requirement legislation [9]. In Australia, DesignCheck [10] was built on the Express Data Manager (EDM) Model Server but, to the best of the authors' knowledge, it has since lost support. The General Services Administration (GSA) in the United States mandates that their project models be checked with rules implemented within SMC [9].

BIM API. While not specifically model-checking tools, BIM editors, such as Autodesk Revit [11] and Graphisoft ArchiCAD [12], provide APIs (the former public while ArchiCAD's requires permission) that allow access to the model's internal structure and object database and therefore, can, in principle, be used for model checking. This requires a high level of programming knowledge even for the simplest checks. To address this challenge, some tools have been developed to perform the same functionality in a visual environment. These include tools such as Autodesk Dynamo [13], which works on the Revit platform, and Rhino Grasshopper [14]. These two tools are both graph-based visual editors that have some scripting available - Dynamo's scripting being in Python rather than C# as the Revit API.

BIMServer [15], an opensource IFC model repository platform, has a model-checking plugin, however, it requires direct coding in JavaScript. The scripts are then linked to the model for execution. This also requires programmatic coding knowledge and a strong understanding of the IFC vocabulary and syntax.

Semantic Web Ontologies. More recently, there has been a conceptual shift in model-checking approaches, given the emergence of semantic-web technologies. Specifically, newer methods have worked with extendable IFC based ontologies of the BIM model to query for design flaws. While this technology can be useful in extending the data schema, the query languages require a steeper learning curve. The basics of this approach are outlined in [16].

Natural Language Processing (NLP). As we mentioned in the introduction, attempts have been made to parse natural-language rules from design handbooks and regulation texts. While such approaches could potentially simplify the rule-creation process, many of the natural-language rules lack the clarity and unambiguity required to be directly parsed without any human intervention or interpretation.

One of the more commonly cited approaches in this vein is that of Hjelseth who used a four-sentence component classification to parse natural language rules, namely Requirement, Applicability, Selection, Exception (RASE) [17] [18]. Another use of NLP has been to identify information from rules that is missing or may need to be added to models [19].

Rule-Checking Languages. The Building Environment Rule and Analysis Language (BERA) [20] was developed as a domain-specific programming language for model checking. The concept is built on providing model-checking capabilities without the need for precise knowledge of general-purpose programming languages [21]. However, the language derives heavily from Java which may be difficult for non-programmers and it is built on the Solibri IFC engine, and therefore is still quite opaque.

Visual Programming Languages (VPL). Some approaches have taken the Rule Languages one step

Natural Language Rule:	**ModelCheck Language:**
Dishwasher must be a minimum of 21" from a corner.	∀ a = IfcDishwasher ϵ ModelObjects ∀ b = IfcCorner ϵ ModelObjects (Distance(a, b)>=21INCH)) True=>Pass; False => Error

Figure 1. An example of a design rule in natural language converted to our rule language. Note that in IFC, there is not type *IfcDishwasher* or *IfcCorner* and the Distance function is not explicitly defined as a relation property. These three elements exemplify the three proposed extensions of our language namely Virtual Objects (*IfcCorner*), implicit geometric object relations (Distance), and expansion of the BIM object type hierarchy (*IfcDishwasher*).

farther by adding a visual component to them, in the same sense that Dynamo is a visual language for Revit's API. This is intended to allow for more complex rules to be created without adding the need to code programming. Check-mate [22] first introduced this as a very simple puzzle-based interface that allowed connecting pieces that together would form a structured rule, however, the expressiveness of this language is limited. The Visual Code Checking Language (VCCL) took a node-based approach, calling it a "white-box" approach with the available nodes to be extendable as the project matures [23], although, to the best of our knowledge, geometric properties and relations cannot be expressed, unless precalculated as properties.

3 Methods

Our methodology for rule checking follows the four-stage process outlined by [9]: (i) Rule interpretation, (ii) Model preparation, (iii) Rule execution, and (iv) Reporting. The following subsections outline our approach to each stage.

3.1 Rule Interpretation

In our work, we have opted to work with a custom, structured rule language approach. As many rules are inherently logic-based and BIM can be viewed as a database for building properties and relationships, our language derives much of its structure from mathematical logical reasoning and database languages, such as Structured Query Language (SQL). As Niemeijer et al. [22] suggested, it is easy to see the similarity between a statement "*For every x in Real Numbers…*" in mathematics with "*For every Window…*" in building regulations. As BIM is a collection of 3D objects, their properties and the relationships among them, we define a model as a set of objects and a set of relationships. Therefore, similar to SQL, our rule language expresses queries on two sets or tables (the FROM element) and determining the result (the SELECT element) of a logical expression (the WHERE element). The exact implementation does not use a specific database query language; we simply use SQL to illustrate our rule language. Figure 1 describes one example of a how a rule can be interpreted from natural language to our proposed rule language.

3.2 Model Preparation

Data in IFC is structured in a highly complex manner as an objects' mesh representation can take the form of extruded solid, Boundary Representation (BREP), or their combinations. This implies that, before the rules can be evaluated, the BIM data must first be transformed into structural objects that support efficient geometric calculations. Similar to [24], our method parses the model into an internal object structure that includes a global triangulated mesh, a local triangulated mesh, and a global bounding box that contains the direction and dimensions of the object in 3D space; a mesh being a series of vertices grouped into sets of three forming triangular boundary faces. Every object with type nested under *IfcElement* is extracted and placed in the set of objects that can be checked.

Once all the IFC objects have been read, our method constructs and adds to the model several different types of Virtual Objects (VOs) which, we define as objects that represent complex, multi-object relationships. By this definition, some VO types are already included in the IFC vocabulary, such as *IfcSpace* and *IfcSite* for example, nested under *IfcSpatialElement*. We extend this list of possible VOs to include *IfcCorner*, as shown in Figure 1, which is the connection between *IfcWall* objects. VOs have geometric bounds and therefore are represented internally much like *IfcElements*. The major difference between the two is that VOs are created based on *IfcElements* and thus depend on them, whereas *IfcElements* have no strict dependence relations.

Properties of objects and object relations are the

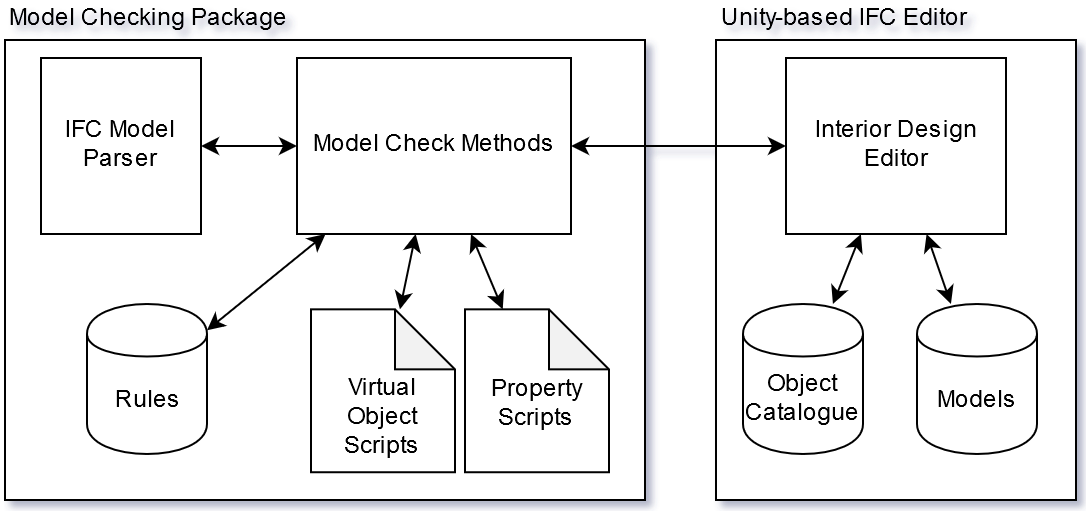

Figure 2. Current implementation of the IFC editor and model checking package.

underlying items being checked by the rules. However, these must be implicitly calculated from the model's geometric data. The geometric properties of objects, such as "Width" and "Height" for example, can be calculated directly on the object's mesh, while geometric-relation properties, such as "Distance" and "Overlap", must be derived from calculations between the two object meshes.

3.3 Rule Execution

All geometric properties and VOs are calculated on a need-to-know basis, thereby intertwining it with the model preparation. As an example, the distance between two objects of a certain type is not calculated unless it is necessary for a rule. Once calculated, it is cached, and can be reused as necessary, until the complete set of rules has been evaluated and the building model-checking is complete.

An interesting challenge is how to archive the computations performed, i.e., the VOs and the relations among objects, in support of the complete model-checking process. In principle, there are two choices: (a) they may be saved with the building model itself, or (b) they may be saved in a separate data structure but with references to the building model.

Should the VOs and properties be saved to the model, it would be necessary to develop a management process to remove the results of individual rule evaluations as the objects to which the rules apply are modified. For instance, if the "Distance" relation property was calculated but the dishwasher has been moved in the new model version, then the original "Distance" property should be removed and recalculated if required.

If the building model editor is capable of flagging the objects that have been modified since the last model check, the model check could recalculate the VOs and properties that depend on those modified objects. This would theoretically expediate the subsequent model checks.

The safer, more conservative, choice is to assume the building model has not been checked previously and that all VOs and properties must be newly calculated and, if existing, then overwritten by the new values. This is the current practice in our prototype, however, we are currently investigating the most efficient way to save and flag changes in our editor.

3.4 Reporting

Finally, all results need to be relayed back to the end user or application. This is returned in the form of an object set, along with the rules that have been evaluated relevant to those objects, and the result of the rule evaluation. While returning all results is important, the reports may also be narrowed down to only the failed rule instances. This allows the client application to parse the result information, graphically display the objects that failed the rule, and display the rule information including the error level of the rule.

4 Discussion

As validation for the proposed research, we have created a prototype model-checking .NET library, as seen in Figure 2.

The library is used by an in-house Unity-based IFC editor of building interiors that is currently capable of

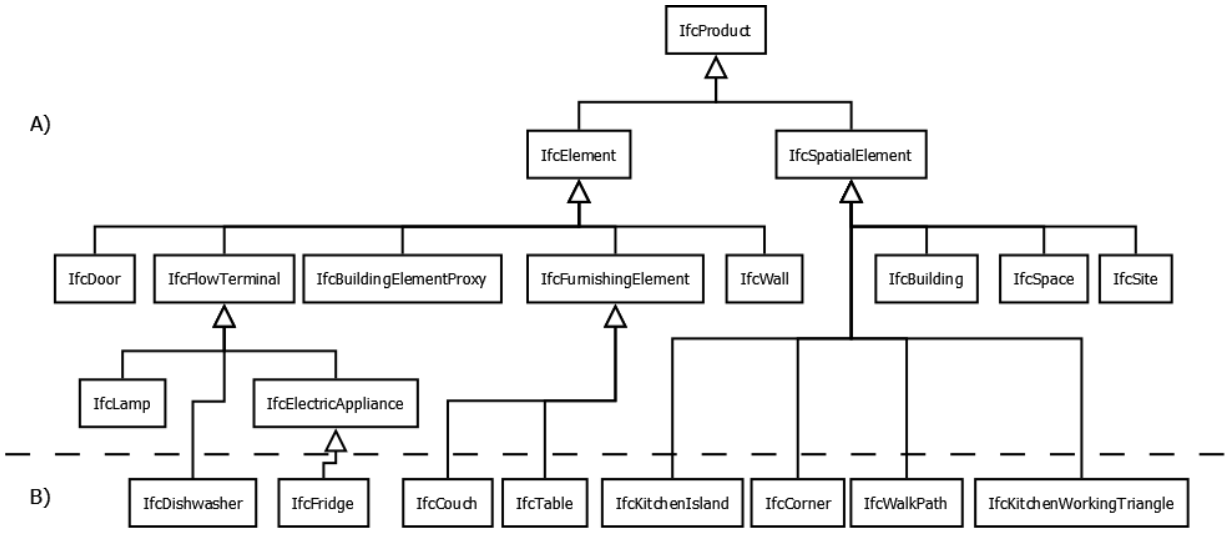

Figure 3. A) IFC object types above the dashed line represent a subset of the current IFC4 schema. B) IFC object types under the dashed line indicate sample extensions to the IFC hierarchy; those under *IfcSpatialElement* representing VOs that would be created implicitly from the existing model *IfcElements*.

reading an IFC file, displaying the model in 3D, and enabling a user to review an object catalogue and add *IfcSystemFurnitureElements* to the IFC model.

The underlying IFC model can be passed to the model-checking method, which is currently configured with an initial rule set, provided by our industrial partner. The rules in this initial set were specified in our language, and subsequently implemented into source code used by the model-checking module. The implementation process is currently manual, but we have developed a library of basic functionalities for computation of VOs and numerous geometric functions (i.e., for calculating object height and width, and distance between objects) and we are currently working on a model-driven method for automatically transforming rules into source code relying on this library.

After execution, the model-checking method returns the results for each of the rules. A rule result includes a pass/fail boolean as well as each instance of the set of objects that were checked against for the particular rule and the result of that instance.

For example, for the rule in Figure 1, an object set instance would be *dishwasher1* and *corner1*, which would have an instance result of pass/fail. The rule result is the collective result of the ANY/ALL/NONE of each object instance and their corresponding instance results. The IFC editor is therefore able to highlight the instances where an object fails a particular rule. The VO meshes are also accessible and can optionally be displayed in the IFC editor.

Since the majority of these rules deal with furnishings and appliances, there is an additional stage in the model preparation that is required but not yet implemented. In addition to restructuring the object representations into our mesh objects, we also believe the object type hierarchy within the IFC schema needs to be extended. Furniture and appliance items typically fall somewhere within either *IfcFurnishingElement*, *IfcFlowTerminal*, or *IfcBuildingElementProxy*, with each of these being the leaf or the second lowest level of the hierarchical tree. While it is possible to add a property to each object that states explicitly the object type (as in the current implementation), we believe more specific IFC types, such as *IfcCouch*, *IfcFridge*, etc. are required. Additionally, this hierarchy should be extendable such that new types defined later should be included. Figure 3 demonstrates a subset of the IFC4 [25] schema with examples of additional object types.

For the purpose of this study, we used the object name to determine automatically whether an object can be categorized, however, the onus is on a more intelligent BIM editor to infer the most specific type of the object, beyond *IfcElement*. It is also imperative that objects do not fall under multiple categories or are compositions of multiple other objects. For instance, difficulties can arise when a collection of objects, such as multiple chairs surrounding a table, are modeled as a single object. Therefore, good modeling practices should be adhered to the largest extent possible.

Other issues encountered included the direction of the objects not always being standardized in IFC, or at least by the BIM editors that export the models. Therefore, relationship properties such as behind and in

front of, which appear frequently in our rules, would occasionally return erroneous values. We see this as an error in the object design, since from an end user viewing the model, this error would not be visually apparent.

Finally, while we acknowledge there are many possible design rules and that they often come in the form of large, text-based documents, we believe that future iterations of these rule documents should be made in tandem with the rule language rules. This would both ensure that text-based rules can in fact be quantified and calculated and that automated rule check results are as the rule creator intended. The use of NLP may help expediate this process but our belief is the onus should be on the rule creator to interpret and test rules as they produce them.

5 Conclusion

In this paper, we have provided a brief introduction to our rule-specification language and a model-checking method able to evaluate rules in this language on IFC building models. Our work aims to connect many concepts put forward in previous model-checking approaches. Several building model-checking approaches exist, each with its own advantages and shortcomings. We therefore believe that providing an extensible framework to enable different rule sets to be expressed and evaluated against different BIM objects should each be supported. Each rule, regardless of method used for its specification (logical, mathematical expressions, or SQL operations on data), should be executable in the same process such that future applications can take advantage of model checking. For instance, we believe model checking can be a useful integration into generative design, which for runtime optimization would require rules sequences to be reprioritized. This will be created in latter iterations of the project and will be implemented as part of the grander scheme of a BIM service framework.

6 Acknowledgements

This work was supported by an NSERC CRD grant entitled “Development of cloud-based collaborative BIM modelling software”.

7 References

[1] Lopez, R., and Love, P. E. Design error costs in construction projects. *Journal of construction engineering and management*, 138(5):585-593, 2011.

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

[3] Solibri. Online: https://www.solibri.com/ Accessed: 30/01/2019.

[4] BuildingSMART. Online: https://www.buildingsmart.org/ Accessed: 30/01/2019.

[5] Statsbygg BIM Manual, Version 1.2.1(SBM1.2.1). Online: https://www.statsbygg.no/files/publikasjoner/manualer/StatsbyggBIM-manual-ver1-2-1eng-2013-12-17.pdf Accessed: 30/01/2019.

[6] Jiang, L., and Leicht, R. M. Automated rule-based constructability checking: Case study of formwork. *Journal of Management in Engineering*, 31(1):A4014004, 2014.

[7] Lee, Y. C., Eastman, C. M., and Lee, J. K. Automated Rule-Based Checking for the Validation of Accessibility and Visibility of a Building Information Model. In *Computing in Civil Engineering 2015*, pages 572-579, 2015.

[8] Getuli, V., Ventura, S. M., Capone, P., and Ciribini, A. L. BIM-based code checking for construction health and safety. *Procedia Engineering*, 196:454-461, 2017.

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

[10] Ding, L., Drogemuller, R., Rosenman, M., Marchant, D., and Gero, J. Automating code checking for building designs-DesignCheck. *Clients Driving Construction Innovation: Moving Ideas to Practice. CRC for Construction Innovation*, pages 1-16, 2006.

[11] Revit. Online: https://www.autodesk.com/products/revit/overview# Accessed: 30/01/2019.

[12] ArchiCAD. Online: https://www.graphisoft.com/archicad/? Accessed: 30/01/2019.

[13] Dynamo. Online: https://www.autodesk.com/products/dynamo-studio/overview Accessed: 30/01/2019.

[14] Grasshopper. Online: https://www.grasshopper3d.com/ Accessed: 30/01/2019.

[15] BIMServer. Online: http://bimserver.org/ Accessed: 30/01/2019.

[16] Pauwels, P., and Zhang, S. Semantic rule-checking for regulation compliance checking: An overview of strategies and approaches. In *32rd international CIB W78 conference*, pages 619-628, Eindhoven, Netherlands, 2015.

[17] Hjelseth, E., and Nisbet, N. Capturing normative

constraints by use of the semantic mark-up RASE methodology. In *Proceedings of CIB W78-W102 Conference*, pages 1-10, 2011.

[18] Hjelseth, E. Converting performance based regulations into computable rules in BIM based model checking software. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM*, pages 461-469, 2012.

[19] Zhang, J., and El-Gohary, N. M. Extending building information models semiautomatically using semantic natural language processing techniques. *Journal of Computing in Civil Engineering*, 30(5):C4016004, 2016.

[20] Lee, J. K. Building environment rule and analysis (BERA) language and its application for evaluating building circulation and spatial. *PhD diss., Georgia Institute of Technology*, 2011.

[21] Lee, J. K., Eastman, C. M., and Lee, Y. C. Implementation of a BIM domain-specific language for the building environment rule and analysis. *Journal of Intelligent & Robotic Systems*, 79(3-4):507-522, 2015.

[22] Niemeijer, R. A., De Vriès, B., and Beetz, J. Check-mate: automatic constraint checking of IFC models. *Managing IT in construction/managing construction for tomorrow*, pages 479-486, 2009.

[23] Preidel, C., and Borrmann, A. Automated code compliance checking based on a visual language and building information modeling. In *Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC)*, Oulu, Finland, 2015.

[24] Solihin, W., Dimyadi, J., Lee, Y. C., Eastman, C., and Amor, R. The Critical Role of the Accessible Data for BIM Based Automated Rule Checking System. In *LC3 2017: Proceedings of the Joint Conference on Computing in Construction (JC3)*, pages 53-60, Heraklion, Greece, 2017.

[25] Industry Foundation Classes Release 4 (IFC4) Documentation. Online: http://www.buildingsmart-tech.org/ifc/IFC4/final/html/ Accessed: 30/01/2019.