2021 European Conference on Computing in Construction
Online eConference
July 26-28, 2021

BIM-kit: AN EXTENDIBLE TOOLKIT FOR REASONING ABOUT BUILDING INFORMATION MODELS

Christoph Sydora and Eleni Stroulia
University of Alberta
Edmonton, Alberta, Canada
{csydora, stroulia}@ualberta.ca

ABSTRACT

Since 2010, research on cloud-based Building Information Modeling (BIM) has been receiving increased attention, motivated by the need to increase productivity through interoperability in the construction industry. To date, IFC is the de-facto standard for data exchange among tools in this domain. However, IFC is too large and not sufficiently specific to effectively support the management of a single building model by multiple tools. This paper proposes BIM-kit, a collaborative BIM platform based on a simple, modular data schema. In this research, we demonstrate how multiple task-specific tools can be used collectively and efficiently in a shared cloud-based BIM environment to support a variety of use cases, across the overall building-design activity.

INTRODUCTION

The construction industry is increasingly being digitized and a variety of tools are being developed to support design-and-construction activities, from building design, to materials procurement, construction project management, interior design, and building usage simulation. Collectively, these tools and the process of constructing digital building models is known today as Building Information Modeling (BIM). Towards enabling the interoperability of these BIM tools, the industry developed the Industry Foundation Classes (IFC) (buildingSMART IFC 2021) interchange format.

IFC is a structured textual format that supports the representation of a BIM model, i.e., most information about a building, including its structural elements, i.e., walls, floors, doors etc., their relations, such as adjacency for example, and their geometries. An essential first step towards interoperability, IFC suffers from a number of shortcomings. First, the complexity of the information required by the potentially many building-construction tools makes the IFC schema quite large. In IFC2x3 (2006), there are 653 entities and 327 types (117 defined types, 164 enumeration types, and 46 select types). By 2015, when IFC4 was released, the numbers had grown to 766 entities and 391 types (Solihin & Eastman 2016). In effect, the IFC representation of a building is a quite verbose textual representation of everything that a tool knows about the building at a specific point in time. Because of the size and complexity of the IFC schema, Model View Definitions (MVDs) were created to filter the schema into subsets of types and entities relevant to different activities. As a result, even the tools that import and export IFC models are unlikely to understand and use all their elements; instead, they only work with the partial subset of IFC data that is covered by their domain-specific MVD, possibly causing ambiguities and inconsistencies in the overall model. Finally, large as the IFC may be, it cannot include all the information required by all the tools that may consume it, since the level of detail required by each tool for each IFC element depends on the tool functionality. As a result, each tool tends to add to it additional proprietary information that is not meaningful to, and gets ignored by, other tools.

As long as one uses IFC as an archival representation of the work-product of a tool, these shortcomings are not particularly problematic. They become much more pronounced, however, when one considers the opportunity of a number of users working on different aspects of a building on their own preferred tools, as envisioned, for example, by cloud-based BIM (Afsari et al. 2016). Current solutions offer shared repositories of IFC building models and a set of APIs through which third-party tools can download the most recent state of the model or upload their modifications to it. As we discussed above, the inconsistencies and ambiguities that each tool introduces as they modify their own MVD-specific model subsets make this visionary workflow practically impossible.

A number of recent publications, which we discuss in detail in the next section, recognize the shortcomings of IFC's one-(big)-size-fits-all approach. In this paper, we describe BIM-kit[1], our research collaborative platform that relies on a different type of model sharing: instead of requiring that each tool exports and consumes a complete IFC model, the shared BIM-kit repository is organized around small cohesive elements, cross-referenced with each other. At the core of this organization are a set of well-defined basic geometry properties and relations, applicable to

[1]BIM-kit code can be found at: https://github.com/csydora/BIMkit

an extendible hierarchy of object types.

As we will demonstrate, this approach supports commonly sought after use cases of cloud-based BIM, such as model-checking applications, building simulation, viewing and editing applications, and automated design, that are at best challenging to achieve using the standard practice of IFC-centric cloud repositories. In fact, there is no demonstration of IFC seamlessly supporting all these tasks. Therefore, BIM-kit is targeted towards both designers that wish to utilize multiple automated cloud-based BIM services, and the developers of BIM services such that standardized and comprehensive models are maintained.

The BIM-kit repository is implemented in MongoDB, a document-centric NoSQL database that naturally supports this kind of representation. To ensure the consistent manipulation of the repository objects, BIM-kit implements a number of APIs to ensure that the objects are correctly updated by the tools integrated with it.

The key elements of the BIM-kit repository are the building models, a catalog of objects which are referenced in the building models, a list of properties and relations, and a taxonomy of object types. To date, BIM-kit integrates six different tools, all of which exchange data with the central repository. Some of them support users to visually experience and manipulate the models and others implement completely automated model-manipulation services. At a high level, the currently available BIM-kit tools are the following:

1. The *Building Model Editor*, implemented in the Unity Game Engine, enables users to select items from a catalog of available and pre-designed objects and place them in the building model.
2. The *Rules Management Service* manages a design-rules repository, which stores building codes in a structured format, such that they can be interpreted and executed on a building model.
3. Three *Rule Editors*, each of which enables users to create and modify design rules and to store them in the Rules Management Service. Each editor offers a different user experience allowing end users to use the editor that best suits their abilities or preferences.
4. The *Model Checking Service* takes as input a set of building code rules from the Rules Management Service and the building model from the BIM-kit repository and returns as output a set of references to model element that relate to the code rules.
5. The *Generative Design Service* takes the functionality of the Model Checking Service one step further: using a set of rules from the Rules Management Service and an initially empty building design from the central BIM-kit repository, it returns a model that includes a set of desired objects, placed in a manner that respects all relevant rules.
6. Finally, the *Model Occupancy Simulator* takes as input a building model, and given a configuration of occupants and their access to the building spaces, it simulates the occupants' activities in the building and returns a set of interesting usage indicators.

The objective of this research is to describe a cloud-based BIM solution that supports a number of use cases, of which including automated design and its supporting model evaluations. We argue that this work makes the following two contributions to the state-of-the-art. First, it puts forward a well-defined, extendible BIM model; the BIM-kit model is compatible with IFC, in that it can import a complete IFC model and create a corresponding set cross-referenced objects. At the same time, because the model consists of a number of distinct objects that are manipulated through well-defined APIs, the model can be more easily shared across different tools. The repository design enables the extension of the model with additional objects, all of which are supported by a core set of geometrical definitions. Second, it demonstrates the usefulness of the above model through the implementation of a variety of interactive and automated tools that, together, cover a wide range of activities that reason about building models. We envision BIM-kit will be a valuable research testbed for building design applications with reduced training overhead.

The remainder of this paper is organized as follows. The next section reviews the related work and highlights the advantages of BIM-kit relative to similar efforts. Next, the manuscript outlines the BIM-kit data model, storage, and access APIs. Following this description, the tools currently available on BIM-kit, their functionalities and their implementation status are reviewed. Finally, we elaborate on our experiments to date with these tools, and conclude with a summary of the lessons learned through the BIM-kit development process, and our plans for the future.

RELATED WORK

`BIMServer` (Beetz et al. 2010) is a well known open source IFC model server, actively maintained and regularly updated. It is a IFC model repository that is capable of performing version control and enforcing access-control rules, checking user credentials against their authority to access the models they request. The associated `bimviews` web app enables users to view the models, and additional functionalities can be created by developing `BIMServer` plugins, such as the example Model Compare and Query Engine Plugins. There have been attempts at developing model-checking plugins for `BIMServer`, but a complete solution has yet to be developed.

There are a few additional examples centered around IFC models such as `BIMCloud` (Das et al. 2014) that focus on the social interaction around the model, and `CloudServerBIM` (Logothetis et al. 2018) that extended the `BIMServer` functionality to include web-based services. By using IFC as the model schema, the above methods face the IFC related development challenges, such as the model inconsistencies due to modification. Autodesk BIM 360 (2021) and GRAPHISOFT BIMcloud (2021) are some example of Cloud-based BIM model servers that do not use IFC but therefore are linked more directly to a single design software.

Our work shares much of the motivation and some implementation decisions with three recent publications that have proposed, or used, MongoDB for implementing repositories of BIM models. Ma & Sacks (2016) use MongoDB to store IFC files, motivated by the need for a dynamic data schema: their system uses the IFC schema but also enables dynamic addition of user-defined properties. The authors have demonstrated the use of their system for the task of reconstructing reinforced concrete beam model from earthquake damaged buildings. Lin et al. (2016) focus on the experience of end users querying massive IFC models for data retrieval and develop a tool that translates natural-language queries to map IFC entities. On the back-end, the tool uses MongoDB to store the IFC data, with Objects, Relations, Types, Geometry, and Relational Objects. This is closely related to our implementation however, we use the platform more broadly, allowing for services and application to modify the model, as opposed to theirs that only adds information to the model for query and retrieval purposes. Jiao et al. (2013) propose a "logically centralised but physically heterogeneous" database where non-geometry data is stored in a relational database and geometric data in a MongoDB database. From a Revit file (Autodesk Revit 2021), the geometry elements are classified, grouped and stored in HOOPS files collectively, with the non-geometrical information being linked via a Globally Unique Identifier (GUID) and stored in the relational database. Our solution, on the otherhand, stores the information collectively and is more intuitive to parse and thus modify.

The most recent and most closely related to our work project is the Building Information Modeling Rule Language Simplified Schema (`BIMRLSS`) Solihin et al. (2020). They argue for a simple model representation to promote accessibility, and believe that at its core, a model should store the "objects, types, their properties, relationships, and their final geometries". Additionally, they cite model-checking as the key motivator to their approach and use a rule-language method for deriving the rules; our rules being tested on interior design, while their rules relating closely to building paths and line of sight. They also consider the benefits of Level-of-Detail (LOD) for the stored model; however, they focus on the accuracy of the geometry representations, where we are more interested in the logical organization of the building model and its constituent elements. `BIMRLSS` and BIM-kit differ in two important ways. First, `BIMRLSS` adopts a traditional relational storage model for BIM data, i.e., non-geometry data, as opposed to our document-centric MongoDB. Second, `BIMRLSS`, similar to `BIMServer`, proposes that additional services should be treated as plugins rather than independent services, which requires a centralized development model of the overall toolkit, and is less extendible than out API-centric approach.

The majority of the previous research does not examine how each of the various design activities use and manipulate the data in the building model. In this work, we look specifically at the data required for each activity and we propose APIs as the mechanism for accessing the various building-model elements in semantically consistent manner.

BIM-kit DATA MODEL & REPOSITORY

Data Model

BIM-kit data is stored in MongoDB as document classes, in BSON, a format similar to JSON. The MongoDB schema contains five core element classes, namely *Model*, *Object*, *Material*, *Type*, and *User*. Client applications "see" the client-side schema, diagrammatically depicted in Figure 1. The difference between these two, as we will outline in more detail below, is that objects in the *Model* that have been included from the object catalog are independent from the *Model* in the MongoDB schema and in MongoDB schema contain multiple Level-of-Detail (LOD) representations.

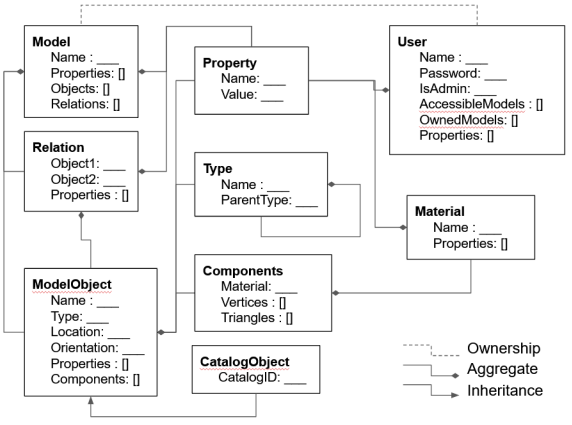

Figure 1: BIM-kit client-side data model.

The *Model* class is the root of the building model, identified by a unique ID and a name. It contains two types of objects, which are the items inside a building: *CatalogObjects* and *ModelObjects*. On the client-side, both will appear to be the same, however, *CatalogOb-*

jects in the MongoDB *Model* are merely references to the elements in the MongoDB object catalog. Thus, on the client side *CatalogObjects* contain the additional *CatalogID* field, but when stored in the MongoDB *Model*, only contain information about where they should be placed in the space, i.e., (*Location* and *Orientation*). *ModelObjects*, on the other hand, are objects that cannot be found in the catalog, either because they have been constructed for the specific building, or because they do not have a fixed geometry but rather one that depends on the geometry of other objects. These would generally include Walls, Floors, and Ceilings, but can also include what we define as Virtual Objects such as Rooms/Spaces, Paths, and Floor Levels. For this reason, *ModelObjects* must store the actual geometrical shape representations in both client side and the MongoDB *Model* as a list of *Components* along with object *Type*. The *Model*, *CatalogObjects*, and *ModelObjects* all have a *Properties* collection associated with them.

MongoDB catalog objects are independent of specific building *Models*; their internal structure is similar to that of the custom *ModelObjects*, since they describe a geometrical shape, basic geometrical properties, and the object's *Type*. However, each MongoDB catalog object may be associated with a number of shape representations (also known as MeshRep), which are a list of *Components* for different LODs. When a model is requested and MongoDB catalog object are reinserted back into the client-side *Model*, the user specifies the LOD those objects in model will be in. This LOD is particularly important for applications that are restricted to the size of the model due to performance limitations. For example, an automated design method may first reason about very coarse objects, such as bounding boxes, for example, and switch to displaying the objects in high resolution when a final output solution is determined.

The idea of separating objects from a catalog from building *Models* is motivated by three reasons. The first is storage efficiency: a single catalog object can be referenced, instead of being cloned, in multiple models. At the same time, computational efficiency is supported by the multiple LOD for each object. The second reason is the need to standardize the object usage as much as possible. Having a catalog can streamline the model-checking process, by reducing the need to recalculate common object geometric properties repeatedly, reducing searches by geometry and eliminating the need for redundant recalculation. Finally, this design decision aligns with objects in the real world, and could potentially increase the relevance of such a tool to product retailers.

In addition to the objects, *Models* also include a list of *Relations* between two relating objects. They contain a reference to the two objects, that can be either *CatalogObjects* or *ModelObjects*, and a set of geometric relations that hold true between the two. Two common property examples would be a relation between two objects with a distance property, or a property that indicates one is an aggregate of another object (Sydora & Stroulia 2019).

The next two element types are meant to standardize the way items are labeled. *Types* define a tree structure, or a taxonomy, meaning each *Type* has a parent *Type*. Ideally, *ModelObjects* and *CatalogObjects* should only use the leaf *Types*, with intermediate *Types* only being used for search.

The *Material* element standardizes the material names, such that rendering applications can expect certain material values and properties for display, while also having a list of properties for physical attributes.

User documents exist in order to restrict model access, by storing ownership and accessibility. For now, only users with administration access can add new *CatalogObjects*, although in the future this can be changed such that *CatalogObjects*, like models, have owners. Additionally, private objects could be used to save *ModelObjects* for each user, thus increasing object reusability.

BIM-kit Repository

The BIM-kit Repository is the service in charge of the storage and retrieval of data to and from the MongoDB store. While the data is stored in five separated document classes, the models and objects being sent to and received from the BIM-kit Repository take client-side schema in Figure 1. This is the where the references to the MongoDB catalog objects in the model are replaced with the actual stored objects, but only using the shape representation LOD defined in the request. Therefore, the BIM-kit Repository receives the model ID and LOD in the request and reconstructs the data to form what is then returned to the client.

The BIM-kit Repository is also responsible for validity in the model. For instance, if a model is uploaded that does not satisfy the structure, or invalid ID references, then the BIM-kit Repository will rectify those mistakes to the best of its ability or else it will not perform the request.

In its current state, the Building Model Editor in BIM-kit is not able to create walls and floors. Therefore, models in BIM-kit initially come from IFC files, exported from other BIM tools and converted to the BIM-kit client side data structure using the Data Conversion tool. Once a model is uploaded, BIM-kit provides a number of services and tools specific to the tasks of stakeholders. A key functionality of the BIM-kit Repository is the ability to reason about the model and add an additional layer of information to the model in the form of properties, relations, and Virtual Objects. This additional information layer can then be further used by other BIM-kit services to form a common semantic basis for performing a va-

riety of tasks. This is achieved through a shared geometrical library which contains methods for adding property, relation, and Virtual Object information to a model.

USE CASES

Figure 2 shows the current state of the BIM-kit architecture and the flow of data between the repository, and integrated services, and applications. The following subsections describe each of the components of the ecosystem around the BIM-kit Repository, and the task(s) they perform.

Building Model Editor

A key functionality for any design tool is a visual editor for users to interact with the design artifact. For building designs, editing is typically done in a 3D environment, and we constructed our editor using the Unity Game Engine (2021) taking advantage of the out-of-the-box features such as 3D rendering (and in future iterations VR and physics). The editor uses the BIMKitApi library, containing methods for sending requests to the BIM-kit Repository, for data retrieval from the BIM-kit Repository. Edits to the model are made locally and only once the model is saved is it uploaded to the BIM-kit Repository. Currently, the model editor is only able to add catalog objects to an existing model.

The model editor will over time add plugins for the accessing external services. We will also implement VR and AR features such as the work done in our previous work - see Sydora & Stroulia (2018).

Rule Management Service

The Rule Management Service is responsible for managing sets of rules, relevant to building designs. It relies on a rule repository, also implemented in MongoDB, accessible to three different editors for users to specify rules.

Rules store all the information needed to compile into executable code that can be invoked by the BIM-kit Model Checking Service. *Rulesets* are collections of references to *Rules*: meaning if a rule changes, all the rulesets that include this rule are automatically updated to point to the latest version of that rule. This ensures that as building-codes evolve or the experts' understanding of ergonomics deepens, changes to individual rules are propagated in all sets that consider this rule as relevant, thus enabling consistency. When a ruleset is requested, the Rule Management Service constructs the ruleset by finding and linking with referenced rules. For the Rule Management Service prototype security is limited to only a username, and save only the users public name within the model. The Rule Management Service provides RESTful API calls for making changes to individual rules and rulesets. Additionally, it performs validity checks to ensure proper authorization and completeness.

As with the BIM-kit Repository, the Rule Management Service has a corresponding client side RMSApi library which provides methods for sending rule requests and parsing the results. Also included is the Rule Administration Application which is an application for managing the Rule Management Service.

Rule Editors

There are currently three Rule Editors (RE) available in BIM-kit for creating rules, to be used for model checking and generative design: a visual editor based on Google Blockly (2021) (BlocklyRE), a textual editor based on a domain-specific structured language (DslRE), and an editor based on natural language processing (NlpRE). All three editors create and edit rule objects in the Rule Management Service, but each one was designed to support a different user-interaction model, to appeal to different types of users with different types of technical background.

The BlocklyRE, as seen in Figure 3 (left), uses puzzle-like pieces, each one corresponding to an element of the rule object schema, to construct the rules. The DslRE is a .NET WindowsForm application that guides users actions by only providing a selectable list of valid options to develop a textual representation of the rules. Finally, the NlpRE enables a user to describe in (a restricted) natural language a rule and automatically parses this description into a rule object, as expected by the Rule Management Service. The user can then make modifications to the parsed rule to better reflect the input text's intended meaning. The comparison of these three editors is left to a future study, however, each has the same overall functionality in creating, editing, and uploading rules.

Each rule editor communicates directly with the BIM-kit Repository to retrieve the available object *Types* so that they can be included in the specification of the objects to which the edited rules may apply.

Model Checking Service

The Model Checking Service is an automated service that can be invoked by any tool in the BIM-kit ecosystem, or by other third-party client applications. The invoking application must provide a user's credentials as access tokens, in order to ensure that the end user of the invoking application is authorized to access the requested building model and validate it against the chosen ruleset. The Model Checking Service retrieves the model from the BIM-kit Repository and the rules from the Rule Management Service, which it then compiles into executable methods. As a result of executing these methods on the building model a check-result is created and returned to the invoking client. The check-result is a list of rule results, which includes a reference to the rule and to the object(s) in the building model that are responsible for the rule result. As an example, in Figure 3 (middle/right) shows

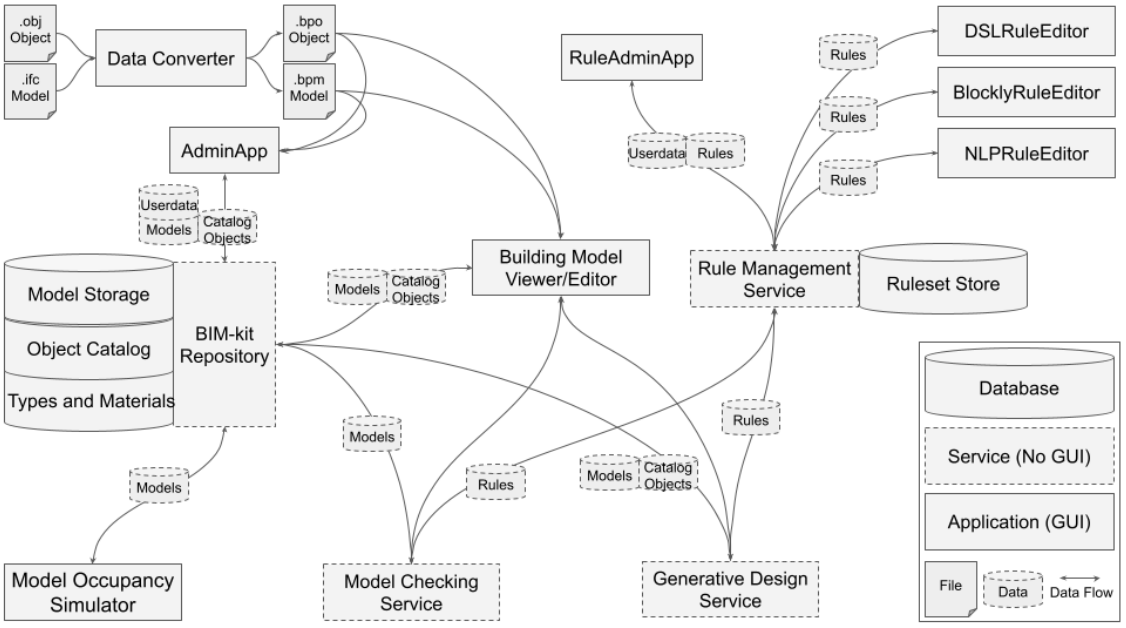

Figure 2: BIM-kit's application and service architecture.

the results from a rule, with the yellow objects being objects relevant to the rule, and the green objects indicating an instance of the rule that have passed the rule check. The interested reader can find an in-depth description of the model-checking method in Sydora & Stroulia (2020).

A key feature of the Model Checking Service is that, in order to evaluate the rules in the context of the building model, it typically calculates the values of a variety of geometrical properties and relations, namely these properties and relations of interest to the invoked rules. These values are stored with the building model, cross-referenced with their associated objects. In this manner, they can be reused by subsequent model checks. If, at any point, the model objects on which these properties and relations depend move or change, the values become obsolete and are, therefore, deleted.

Generative Design Service

Relying on the Model Checking Service, we also developed an automated Generative Design Services that automates the placement of BIM-kit catalog objects in a building model such that the placements validate all the relevant design rules. As with the Model Checking Service, the Generative Design Service acts as a proxy for the client application, and also accesses the model and rules from the BIM-kit Repository and Rule Management Service. In addition, the end user provides as input a list of desired objects from the MongoDB catalog to be included in the building model.

The Generative Design Service included in the BIM-kit uses a variant of the rule-based generation from Sydora & Stroulia (2020). The method uses domain knowledge to determine a series of possible placement locations. It then performs a greedy search of the placement configurations that results in an locally maximum rule compliance score. When a configuration solution is found, a copy of the model with the object placement configuration is created by the Generative Design Service and shared with the user. The user can then save the configuration over the existing model or discard the Generative Design Service solution.

Model Occupancy Simulator

In 2014, Wong et al. (2014) reported that very few CloudBIM projects focused on operation and facility management. In addition to design, building models can be used to manage buildings and simulate activity. In that vein, we recently developed a prototype building-occupancy simulator, based on BIM-kit models. The simulator uses the model data to construct a set of spaces and paths that are accessible to the simulation agents. In our case, the goal was to simulate the risks associated with an infectious pathogen under different building operation scenarios, such as restricting room or floor access, or limiting building occupancy capacity. More generally, however, the simulator acts as an additional evaluation in which time is a factor as elements of the model and the use of the spaces are dynamically changing due to the movements of building occupants.

The simulator is a Unity application that accesses a building model from the BIM-kit Repository and renders the scene, using the same methodology as the Building Model Editor. A navigation mesh (NavMesh), which represents the walkable areas of the building is constructed based on the floor components in the model. Paths can be generated by a combination of the NavMesh and adjacency relations

Figure 3: (Left) Blockly Rule Editor. (Middle/Right) BIM-kit Model Check extension in the BIM-kit Building Model Editor.

extracted from the model, as the NavMesh alone is unable to recognise level-to-level transitions such as elevators. Finally, the simulator takes as input a series of time-stamped agent activities, generated by a special-purpose agent-trace generation tool given a start location and an end location, and simulates the agent's behaviour. During the simulation, the simulator records a variety of interesting indicators, including agent interactions, and occupancy levels at the individual room and zone levels.

DISCUSSION

Advancing Interoperability

BIM-kit is conceived to address the challenge of maintaining a consistent representation of a building model, while at the same time, making it available to a variety of tools for different types of reasoning and design activities. This is a primary problem with IFC, which often times is referred to as a "view only" version of all building information; modifications to an IFC file have to be requested to the designer rather then done directly on the model. On the other hand, BIM-kit allows direct modification of the model, ensuring its integrity through its well-defined modular schema. At the same time, the BIM-kit Repository can ingest and export IFC models, thus ensuring a degree of interoperability with current IFC-based software.

Compared with previous cloub-based BIM approaches, the underlying BIM-kit schema is concise in its representation, but as demonstrated, not limited to a single application and use case.

Semantic Modeling

The underlying semantic enrichment and rule-checking follows much of the same motivations as the work of Pauwels & Zhang (2015), Sacks et al. (2017), Hagedorn & König (2020). BIM-kit has been designed with the expectation that models have little to no information outside of geometrical information. Therefore, the power of BIM-kitis to support the enrichment of the models with semantically meaningful information, such as, for example, the evaluation of interesting geometric relationships between model objects. Such relationships, e.g., alignment, distance in different dimensions, etc, are useful to users exploring the models through the editor, and necessary to the model-checking service evaluating the rulesets to which the model may adhere.

In BIM-kit, the Rule Management Service, the Rule Editors and the Model Checking Service, provide a key functionality, essential for building-model management, namely the definition of logical and geometrical constraints and their validation against building models. This set of tools, in effect, support the specification of operational semantics behind the model representation.

CONCLUSION

In this paper, we discussed our work on BIM-kit, a set of tools supporting a number of reasoning tasks about building models. The BIM-kit set of federated tools relies on a central repository of building models, represented in a well-defined modular and extendible schema.

The BIM-kit schema covers the same information as IFC but is fundamentally different from IFC because it is modular, i.e., it defines a set of simple, highly cohesive classes, cross-referenced with each other. The modular nature of building representation in BIM-kit supports the management of the overall consistency of the models: because the repository contents are manipulated through well-defined APIs, building models can be more easily shared across different tools. Also because of its modularity, the BIM-kit schema supports extendibility. At the same time, the BIM-kit model is compatible with IFC, in that it can import a complete IFC model and create a corresponding set cross-referenced objects, and it can export an IFC model by collecting the relevant classes corresponding to a single model.

The variety of tools we have developed around the BIM-kit Repository constitutes strong and persuasive evidence on the effectiveness of the underlying representation. The six interactive and automated tools around the BIM-kit Repository cover a wide range of design activities, across the building lifecycle, from interactive manipulation, to rule-based validation, to automatic generation of design alternatives, to usage simulation.

ACKNOWLEDGEMENTS

This research was supported by the Natural Sciences and Engineering Research Council (NSERC). Special thanks to Noah Burghardt and Samuel Jaeger for their aid in development.

References

Afsari, K., Eastman, C. M. & Shelden, D. R. (2016), Cloud-based bim data transmission: current status and challenges, in 'Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC)'.

Autodesk BIM 360 (2021). Accessed: January 17, 2021.
URL: https://www.autodesk.com/bim-360/

Autodesk Revit (2021). Accessed: January 17, 2021.
URL: https://www.autodesk.com/products/revit/overview

Beetz, J., van Berlo, L., de Laat, R. & van den Helm, P. (2010), Bimserver. org–an open source ifc model server, in 'Proceedings of the 27th CIP W78 conference'.

buildingSMART IFC (2021). Accessed: January 17, 2021.
URL: https://technical.buildingsmart.org/standards/ifc

Das, M., Cheng, J. C. & Shiv Kumar, S. (2014), Bimcloud: a distributed cloud-based social bim framework for project collaboration, in 'International Conference on Computing in Civil and Building Engineering', pp. 41–48.

Google Blockly (2021). Accessed: January 17, 2021.
URL: https://developers.google.com/blockly

GRAPHISOFT BIMcloud (2021). Accessed: January 17, 2021.
URL: https://graphisoft.com/solutions/products/bimcloud

Hagedorn, P. & König, M. (2020), Rule-based semantic validation for standardized linked building models, in 'International Conference on Computing in Civil and Building Engineering', pp. 772–787.

Jiao, Y., Wang, Y., Zhang, S., Li, Y., Yang, B. & Yuan, L. (2013), 'A cloud approach to unified lifecycle data management in architecture, engineering, construction and facilities management: Integrating bims and sns', Advanced Engineering Informatics **27**(2), 173–188.

Lin, J.-R., Hu, Z.-Z., Zhang, J.-P. & Yu, F.-Q. (2016), 'A natural-language-based approach to intelligent data retrieval and representation for cloud bim', Computer-Aided Civil and Infrastructure Engineering **31**(1), 18–33.

Logothetis, S., Karachaliou, E., Valari, E. & Stylianidis, E. (2018), Open source cloud-based technologies for bim, in 'International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences', Vol. 42–2, pp. 607–614.

Ma, L. & Sacks, R. (2016), A cloud-based bim platform for information collaboration, in '33rd International Symposium on Automation and Robotics in Construction (IAARC)', pp. 581–589.

Pauwels, P. & Zhang, S. (2015), Semantic rule-checking for regulation compliance checking: An overview of strategies and approaches, in '32nd international CIB W78 conference'.

Sacks, R., Ma, L., Yosef, R., Borrmann, A., Daum, S. & Kattel, U. (2017), 'Semantic enrichment for building information modeling: Procedure for compiling inference rules and operators for complex geometry', Journal of Computing in Civil Engineering **31**(6), 04017062.

Solihin, W., Dimyadi, J., Lee, Y.-C., Eastman, C. & Amor, R. (2020), 'Simplified schema queries for supporting bim-based rule-checking applications', Automation in Construction **117**, 103248.

Solihin, W. & Eastman, C. (2016), A simplified bim model server on a big data platform, in 'Proceedings of the 33rd CIB W78 Conference'.

Sydora, C. & Stroulia, E. (2018), Augmented reality on building information models, in '9th International Conference on Information, Intelligence, Systems and Applications (IISA)', pp. 1–4.

Sydora, C. & Stroulia, E. (2019), Towards rule-based model checking of building information models, in 'Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC)', Vol. 36, pp. 1327–1333.

Sydora, C. & Stroulia, E. (2020), 'Rule-based compliance checking and generative design for building interiors using bim', Automation in Construction **120**, 103368.

Unity Game Engine (2021). Accessed: January 17, 2021.
URL: https://unity.com/

Wong, J., Wang, X., Li, H., Chan, G. & Li, H. (2014), 'A review of cloud-based bim technology in the construction sector', Journal of Information Technology in Construction **19**, 281–291.