A formal Approach for Interfaces and Requirements of Smart Objects in Building Models

Dr.-Ing. Jakob Kirchner, jakob.kirchner@tu-berlin.de
Fachgebiet Bauinformatik, Technische Universität Berlin, Germany

Prof. Dr.-Ing. Wolfgang Huhnt, wolfgang.huhnt@tu-berlin.de
Chair of Fachgebiet Bauinformatik, Technische Universität Berlin, Germany

Abstract
Smart objects represent building components in digital models and can draw conclusions from their situation and behave to their current role in the model. The advantage of using smart objects is the modularized implementation of model checks and an intended behavior. In this paper the analysis of the requirements for the used algorithms in smart objects is used to propose a formal approach of declaring interfaces for smart objects in building models. This covers mandatory requirements and requirements for knowledge-based model checking or algorithms that control the behavior. The difference between geometry and properties including the related problems is presented. An application is the automatic query for fitting objects from object libraries according to user-selected interfaces. This approach is intended to extend the current implemented models in software products which are limited by predefined templates and sets of properties.

Keywords: smart objects, parametric modeling, knowledge-based engineering

1 Introduction
The introduction of BIM (Building Information Modeling) requires CAE (Computer Aided Engineering) software systems which allow the creation, enhancement and modification of building models. According to the object-oriented paradigm these models are divided into objects creating an encapsulation of properties and behavior. These objects usually correspond to building components and represent their geometry, semantics and properties. They are also part of object-interrelationships. When properties are used to drive the change of an object they can be called *parameters*. The paradigm of parametric modeling extended by checking rules leads to the idea of so-called *smart objects* (Hjelseth 2016). These checking rules – implemented as algorithms – can be regarded as distributed or modularized knowledge. This corresponds to classification of the principles of 3D modeling in CAD (Computer Aided Design) systems whereas "Knowledge-based CAD [includes the] ability to draw conclusions from the current design situation (geometrical and also background information)" (Verein Deutscher Ingenieure 2009, p.13). Knowledge-based CAD is of a higher level than parametric or feature-based CAD. Current modeling software for BIM already features the idea of smart objects, but limits the possibilities and the modeling domain to predefined object types. This makes it very complicated and sometimes impossible to take advantage of smart objects' strength consisting of modularized knowledge for model checks and other workflows.

2 Method
This research is based on the concept of smart objects as parametric CAD-objects consisting of properties and geometry which are enriched with knowledge as algorithmic rules for checking

input data. It includes also the treatment of special cases that can occur because of dependencies on other objects or properties of the building model as a whole.

In this paper the current state of model checking in BIM models is shortly summarized and is used to classify the possibilities of smart objects. The internal structure of smart objects is analyzed and geometrical and non-geometrical data is treated separately, because they are of a different nature and involve different problems. The state-of-the-art approaches in selected software products and their limits are used to explain the current lack of a formalized way of describing the required and the supplied data of objects in building models. An example is introduced which includes a useful set of rules of a typical smart object. The required data to evaluate these rules is identified. Affected objects which can supply matching data are also included. Certain elements are proposed which can be used to formalize the description of interfaces between smart and other objects.

3 Review of Literature and State-of-the-Art

3.1 Model checking in BIM

The automated rule-based checking of digital building models is a promising approach for increasing the effort of these models (Eastman et al. 2009). The idea is to use algorithmic checks for compliance of the model to modeling conventions, building codes or regulations, project requirements, design decisions or data completeness, etc. (Solihin & Eastman 2015). As available software products like *Solibri Model Checker* require a "high level of customization" (Solihin et al. 2020) there is an ongoing research with efforts to formalize and optimize the checking process. The main parts of the checking process are identified as (Eastman et al. 2009, Solihin et al. 2020):

- Translating the rules from natural language to checking rules in a formal language
- Preprocessing the building model data
- Execution of checking rules

Preprocessing the building model data is necessary as there usually is some explicit data missing or auxiliary data structures are required to allow fast and complete queries in the checking rules. The available approaches for querying rely on the open IFC data model as source for their custom data structures or use the IFC counterpart: the ontology ifcOWL (Zhang et al. 2018).

Solihin & Eastman (2015) classify the rules according to their required data. The complexity increases with each level:

- Class 1 rules require explicit available data like attributes and relations.
- Class 2 rules require derived values which can be calculated without additional data structures.
- Class 3 rules require extended data structures like geometry distance checking.
- Class 4 rules include a "proof of solution" which sets the focus on computing and presenting solutions in the case of a failed compliance.

Rule-based checking is usually not executed instantly because of its large computational demand. Consequently, it is embedded in an iterating process of modeling/fixing and checking. Although *Class 1* and *Class 2* rules might be validated instantly whenever their input changes, they are usually part of the large collection of rules which is checked in each step of the iteration. *Class 4* rules are not widely available apart from a few examples for instance the safety checking of construction models including an automatic generation if needed, described by Zhang et al. (2013).

3.2 Smart objects in BIM

Instead of validating and reporting "Fail" or "Pass" in automated rule-based checking in a global model context (Eastman et al. 2009) smart objects usually work in a local model context. Smart objects differ from the general approaches of automated rule-based checking as they only include rules they are clearly and exclusively responsible for and react instantly to any changes. The term *smart object* is used to describe its role and enhanced abilities. Smart objects can use geometric or non-geometric data for evaluating rules and respond to changes of the referenced data. The goal is achieving the integrity of the part of model the smart object is responsible for. This matches

the goal of automatic rule-based checking of building models: the creation of a digital representation of the building and its components which is correct and realizable according to codes, standards and the modeler's design intentions. Smart objects are one concept to fulfil this goal by implementing the checking rules as part of the domain knowledge (Hjelseth 2016). It's obvious to implement those checking rules in modularized software components which are equivalent to the digital building components. Typical rules are based on manuals and codes of the represented building component. As rules may conflict depending on the current state it's important to provide analytical methods to identify and resolve the conflicts. This is essential because only a conflict-free state guarantees the integrity of a smart object. There are two strategies to keep the integrity of smart objects: The *adaption strategy* and the *feedback strategy*.

Ibrahim & Krawczyk (2003) define a smart object which is "[…] keeping its integrity as a unit and maintaining its relations to other objects". This is the adaption strategy which promises the biggest advantage but is usually the more complicated one. It includes the smart object's mechanism of changing itself to adapt to a new model state. This makes it a parametric smart object. It differs from ordinary parametric objects provided in CAD software in containing more complex and a larger number of rules. These rules are used for checking for violations of the limits set by the domain specific requirements.

The feedback strategy differs as it does not require the smart object to change its state. It simply delivers a feedback message if the smart object cannot feasibly adapt or identifies an invalid state of the model according to its rules. The content of the message can be used to reassure the integrity by prompting the user to adjust the model.

Both strategies require access to different data, but smart objects usually include rules of Class 1, Class 2 and in the case of the adaption strategy locally limited rules of Class 4. To understand the type of data which has to be accessible by the rules, it's necessary to take two different views: The view on geometry because the object is a CAD-object, and the domain-knowledge view because it represents building component possibly part of the real world.

3.3 The view on geometry

The view on geometry is driven by solid modeling in 3-dimension space (Mäntylä 1988) and the requirements for buildable components. The relevant paradigms for solid modeling are (Strout 2006):

- the explicit representation by topological objects of the boundary of the solid called b*oundary representation* (b-rep or BREP)
- the implicit representation by combining primitive geometry objects with Boolean operators in a binary tree called *Constructive Solid Geometry* (CSG)
- the implicit representation by using 2-dimensional shapes and extruding, sweeping and revolving them by vectors or along curves. This is usually referred to as *general sweeping*.
- Other paradigms like cell decomposition are not widely used for engineering applications.

As all paradigms have got advantages over one another, combining the paradigms is state-of the art. That means base objects in an explicit description get used in implicit descriptions like sweeps or Boolean operations. This is used to compute and update an explicit model. It's strongly connected to the idea of parametric modeling where parameters can drive the change of the geometry whenever the assigned values of parameters get changed. The resulting visible topology objects shaping a solid are Vertex, Edge and Face (ISO 2003). Referencing Vertices, Edges or Faces of already created solids – in this case as representations of building components – makes it possible to create a building model through adding object by object. For building models – assuming we would know the process of erecting the real building in the phase of creating the model – connecting each object like they are erected in reality would be a possible solution. This leads to the idea of history-based models, where each user operation including the creation of model objects is stored. The underlying data structure is a list or a tree of commands. It depends on the used modeling software whether the data structure is exposed to the user or not. Whenever a change to the model is committed, the occurrence of the change in the underlying data structure is the starting point for a recomputation of the affected part of the history.

The described procedure for representing and updating geometry is the state-of-the-art in current software products and forms the skeleton for almost every parametric modeling software. Lee et al. (2006) developed the *Building Object Behavior* (BOB), a formalized graphical description of how the geometry of building objects should behave in case of parametric or relational changes. They define "intelligence" of objects as a behavior according to domain knowledge. Lee et al. focus on geometric parametric descriptions and do not use non-geometric parameters. An extension to BOB was proposed by Cavieres et al. (2011). Smart objects do not necessarily need to have a parametric geometry description, but this increases the advantage of their use as it's the main point of the *adaption strategy*.

3.4 The domain-knowledge view

Extending the geometric description of smart objects and how their geometry is linked in history-based models by domain-knowledge leads to non-geometric properties. Adding non-geometric properties – that means they do not describe any part of the geometry – enriches objects with data. In CAD software this is often referred to as *feature* (Shah 1995). A feature (-object) does not have to be necessarily a representation of a real entity.

Domain-knowledge can be used to constrain the feasible values of geometric and non-geometric properties. This allows an instant feedback or even (semi-)automatic preservation of integrity by including a certain rule or constraint. This can be implemented by disallowing invalid values or using a system of rule-based warning messages (Singh 2015). An easier approach is the delivery of predefined value combinations for parameters as catalogs. These catalogs include valid values, but do not necessarily include the checking logic whether these values are valid in the context of the current role and situation of the object.

Fischer (2006) introduces the concept of making building components "[…] self-aware of their functional relationships to systems of one or several disciplines […]". This means the implemented knowledge is used to check automatically in the current context of the virtual building model. This is used to fulfill the goal of better design decisions. As a consequence, the smart object needs access to data from its neighboring objects or larger parts or domains of its hosting building model, e.g. spatial constraints (Chen et al. 2016).

3.5 Smart objects in current software

Current modeling software supports the creation of parametric objects with a certain kind of knowledge. This is usually very limited. In Autodesk Revit this is called "Loadable Families". The accessed data and referenceable geometry from neighboring objects or the building model is very limited and defined by given template files (Autodesk 2021). It's not possible to create new customized template files with different definitions of accessible data.

In Archicad 24 by GRAPHISOFT the objects described by GDL (Geometric Description Language) have limited access to model data, too. A predefined set of so called "Global-Variables" can be used to get preferences from the model or a very small subset of objects (GRAPHISOFT 2020, p. 335-437), but they cannot be customized or extended.

3.6 The lack of interfaces

As current software already defines some interfaces for smart objects a fully customizable definition of interfaces is not yet possible. This leads to the current situation, that the use of smart objects is limited and less expressive, because the lacking access to data needed for complex rules. Interfaces would make it possible to identify required and provided capabilities of smart objects if it is used comprehensively and consequently.

4 Towards the interfaces

4.1 Vision for smart objects

Modeling software supporting the creation of smart objects and the possibility to define what smart objects require opens the door for a new perspective and changes the dealing with objects in a building model. It would be possible to model fast and get an instant feedback, if the base integrity of the design is violated. This requires a library of prepared smart objects which are

designed to fit and work together. Creating and maintaining such a library is only reasonable if it's highly reused in different projects. The idea assumes that each object can provide and require the data it needs to behave correctly. This is realized by deriving the required data from the implemented rules of a smart object.

This consequently leads to a model where smart objects are like software modules. They are registered and integrated in the building model like modules of a software application. Each smart object would carry as many rules as needed for the goals of the model. A set of rules might be grouped by their source, functional purpose or domain. Each set would describe a certain capability of the smart object. This structure makes the rules comprehensible in the case of unresolvable states. Unresolvable states occur if the smart object cannot adapt to the new model state because the geometric constraint system can't be resolved or checking rules fail. Grouping allows a selective softening or even deactivation of certain ruleset. This is essential for analyzing the occurring conflicts.

If each smart object defines the data it might use in its algorithms and also each object defines which data it can provide to other objects, it would be possible to automatically retrieve a list of fitting objects from the library, whenever an open connection point is selected.

4.2 An example

To clarify the needed input for algorithms included in smart objects, an example is presented on using simple checking rules for building components.

Anchor channels can be embedded into components made of concrete for instance walls to provide mounting for other components as can be seen in Figure 1.

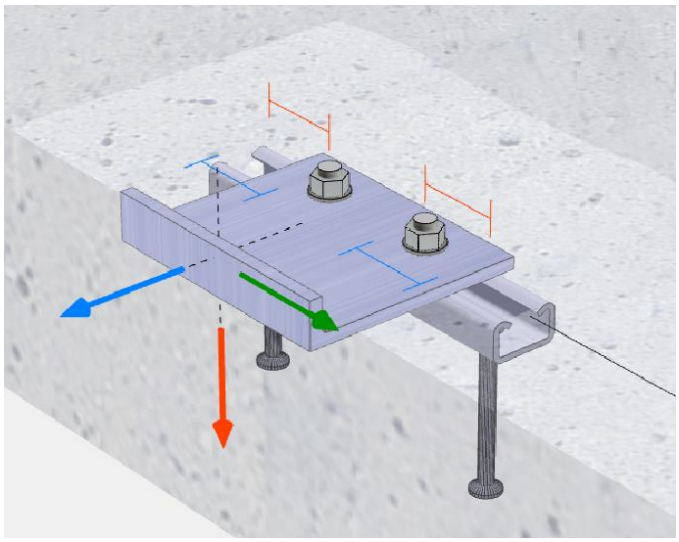

Figure 1. An anchor channel by JORDAHL® with a mounted plate (picture with a friendly permission of JORDAHL GmbH).

As a parametric object, it can be placed on a wall by defining a start and an end point of the anchor channel on a planar face of the wall. These points represent the geometric connection points. The geometry of the anchor channel is parametric and creates the anchors based on the number of anchors and the first and the last anchor's position equidistantly. To verify the minimum distance between the anchors the anchor channel object is enriched with additional checks. For example, the minimum distance between the anchors depends inter alia on the strength class of concrete used in the hosting component. An example of an algorithm according to the anchor channel's official approval codes is included in Figure 2. To execute the algorithm the access to the current distance of the anchors is needed which can be established internally. The access to the property for the strength class of concrete of the other object is the difficult one. To coordinate the access, the property needs to be globally identifiable and has to include a suitable datatype. Whenever the concrete strength class changes or the driving geometric connection points are modified, the allowed minimum distance has to be updated and used for a check of the current geometry. If the

check identifies a violation of the minimum distance the output is a simple warning or error message. It also might be used to mark the object as invalid for the current model state.

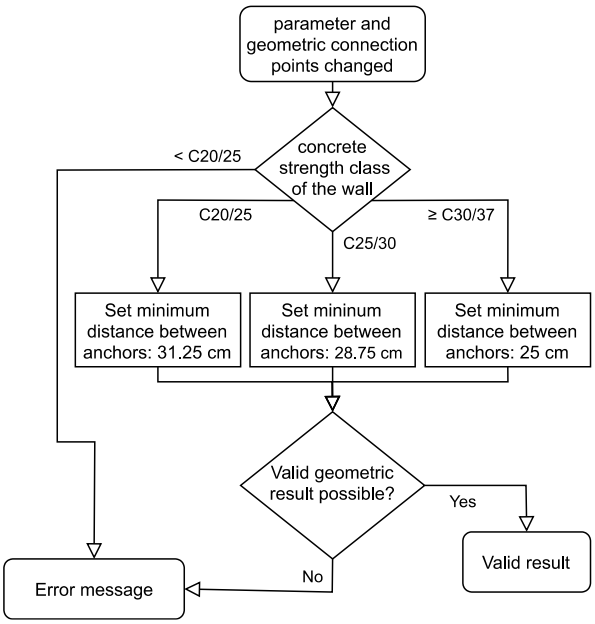

Figure 2. Flowchart of a rule checking algorithm for minimum distance of anchors in dependency of the concrete strength class of the hosting component made of concrete. This is a simplified example based on the design rules in JORDAHL GmbH (2020).

5 Formal description of interfaces

5.1 Basic concept

The basic idea for modeling interfaces of smart objects is the classification into a providing and a requiring side. That means each object declares the properties and geometry objects – geometry objects also include local coordinate systems – it needs to exist and to work properly. Additionally, it can provide access to its properties or geometric objects for other objects.

Each smart object has requirements which are mandatory. Without fulfilling them the smart object can't exist. That means the data is missing to compute a simple valid state for its geometry; its geometric shape is indefinite. For example, an anchor channel needs a provided face of a hosting component where it can be placed. Further requirements are optional, because they are used to activate the "smartness" of the object by establishing the needed access for checking rules.

Checking rules sharing a common goal can be grouped by their required access and form a *port* which aggregates the requirements. The mandatory requirements can also be grouped into a mandatory port. Introducing ports makes it possible to identify what is needed to activate a single capability.

Ports on the providing side represent the access to all properties and geometric objects a smart object is able to share. Additional ports may provide access to unique connectors of the represented components, e.g. sockets for electrical connections, prepared mounting for door handles, etc. An example is shown in Figure 3. The used notation is based on the Ball-and-socket notation for `ConnectableElement` in UML 2.5.1 (Object Management Group 2017). `ConnectableElement` is a suitable modeling representation, because it already includes the concept of ports and interfaces.

Following the concept of parametric modeling, as soon as an object is integrated in the model and plugged into a fitting interface a *binding* is established. Whenever a change is induced to a property or geometry all dependent properties or geometry objects get triggered because of direct or transitive bindings. Smart objects have to update their state and reapply the checking

rules using the model's current state. The creation of bindings assumes compatible interfaces on both sides.

Bindings for properties are always stable because only the change of their value is regarded as an appropriate change. That means the referenced property in a binding never dissolves or changes its underlying structure. Instead, referenced geometry objects in bindings may break or need a different concept.

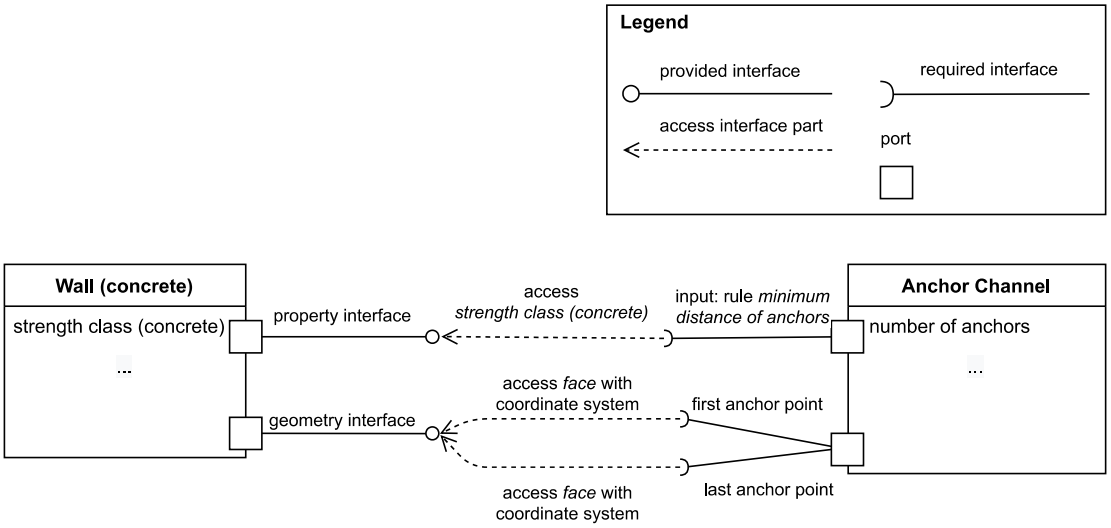

Figure 3. Example of a smart object *anchor channel* and its ports bound to a hosting object *Wall (concrete)* using provided ports.

5.2 Persistence of referenced geometry

Because geometry objects may be removed or split, their references in bindings may dissolve or become ambiguous. Usually implicitly represented geometry cause these problems more often than explicitly represented. It forces a decision how dependent objects should be handled. In CAD models this is called the problem of *persistent naming* (Stroud 2006, p. 382). This problem cannot be solved by the proposed approach. But there is the chance to avoid the problem in some cases or at least give a feedback to the user.

Each object has a port that provides access to the object's available geometry objects. Other objects can establish a binding to these geometry objects. Whenever a change occurs which triggers a topological change the set of provided geometry objects will change. If semantically equal geometry objects get created, they can be offered to smart objects whose bindings just have broken. Either the smart objects include an implemented behavior to automatically choose a new reference or the user can be prompted to choose one. At least a warning message has to be presented informing about the lost binding.

Temporarily freezing a binding and delivering a warning message is also applicable in other inconsistent states for instance an unresolvable geometric constraint system.

5.3 Identification

For the process of integrating a smart object into the model and establishing the bindings it's necessary to make properties and geometry objects identifiable. Of course, all bindings can be set manually, but an automatic process is more effective. Assuming a smart object is added to a building model, setting all bindings individually is a time-consuming task. Instead, selecting other objects aggregates the bindings to the properties and geometry objects of the selected ones. For example, an anchor channel is added by mounting it to a wall and additionally binds to certain needed properties for the checking algorithms. The type of the property needs to have an identifier to enable the automatic selection for the binding. This can be implemented for instance using data templates defined in ISO (2020). This also ensures matching data types for bindings.

So far there is no equivalent formal approach available for identifying geometry objects as they are usually part of the geometry kernel in modeling software without added semantics by knowledge. This is linked with the mentioned *persistence of referenced geometry*. Explicit representations can get a manual classification and identification of the geometry objects which would also be time consuming. A promising approach is based on classification rules. This is also necessary to automatically detect a matching geometry object for instance when docking heating pipes to each other or to predefined connectors.

6 Conclusion and outlook

This paper gives an overview of the current state of smart objects in digital building models and introduces a formal approach for declaring interfaces for smart objects. Current software products only support a limited set of interfaces. They are not customizable and extendable. But this is a necessary step in using the knowledge, integrated in smart objects. Typical use cases are model checking and reactions to modifications which have an effect on the object. The approach using ports would introduce a more intuitive way of modeling by adding more and more smart objects to the model and binding them to each other. It also supports the activation of a smart objects capabilities as soon as the required data can be provided and the bindings have been established. This would also include the support of automatically querying libraries of smart objects for fitting objects whenever an open port is selected. It would also be possible to check which data is missing for selected capabilities.

The next step is a reference implementation. An implementation for an existing software product like Autodesk Revit is partly possible, but a new implementation is more promising. Existing software would have to support a separated level of logic, although it's usually not possible to extend its core features. In a first step the focus would have to lie on properties rather than on the geometry, because the properties and their bindings do not include problems like the mentioned *persistence of referenced geometry*.

The combination of interfaces for smart objects with data dictionaries like the buildingSMART Data Dictionary (bSDD) is a promising approach. Using the identification and template systems of data dictionaries, they can represent the base for modeling smart objects using a common pool of properties.

In this paper the assumption is made that smart objects can only bind to other objects in the model. An extension would be the binding and access to global model and software features. This might include extra query paths, for instance a smart object representing an engine measuring the distance to walls with the purpose of checking the free space for cooling. It would allow *Class 3* rules to be included in smart objects and is connected to the approaches of querying the building model as described by Zhang et al. (2018) or Solihin et al. (2020). Another example are smart objects representing windows prepared for higher wind speeds which can query the model for the vertical distance to the ground and use it for the computation of the wind speeds and dependent checking algorithms. Therefore, an implementation is needed which presumably will be a completely new modeling software.

References

Autodesk (2021). *About Family Templates*. Viewed 2021-04-13, https://help.autodesk.com/view/RVT/2021/ENU/?guid=GUID-E36987A9-A68F-4121-A391-907306BAA60A.

Cavieres, A., Gentry, R. & Al-Haddad, T. (2011). Knowledge-Based Parametric Tools for Concrete Masonry Walls: Conceptual Design and Preliminary Structural Analysis. *Automation in Construction, Selected papers from the 26th ISARC 2009*. 20 (6). pp. 716-728. DOI:10.1016/j.autcon.2011.01.003

Chen, S. Y., Lok, K. & Jeng, T. (2016). Smart BIM Object for Design Intelligence. *Proc. of the 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016)*, The University of Melbourne, Melbourne, Australia, 30th March – 2nd April 2016. pp. 457-466.

Eastman, C., Lee, J., Jeong, Y. & Lee, J. (2009). Automatic rule-based checking of building designs. *Automation in Construction*. 18 (8). pp. 1011-1033. DOI:10.1016/J.AUTCON.2009.07.002

Fischer, M. (2006). Formalizing Construction Knowledge for Concurrent Performance-Based Design. *Intelligent Computing in Engineering and Architecture*, 13th EG-ICE Workshop 2006, Ascona, Switzerland. pp. 186–205. DOI:10.1007/11888598_20

GRAPHISOFT (2020). *GDL Reference Guide (Archicad 24)*. https://help.graphisoft.com/AC/24/SPA/GDL.pdf.

Hjelseth, E. (2016). Classification of BIM-Based Model Checking Concepts. *Journal of Information Technology in Construction (ITcon)*. 21 (23). pp. 354-369.

Ibrahim, M. & Krawczyk, R. (2003). The Level of Knowledge of CAD Objects within the Building Information Model. *Proc. of the 2003 Annual Conference of the Association for Computer Aided Design in Architecture*. Indianapolis, IN, USA, October 24th-27th 2003. pp. 173-177.

ISO (2003). *Industrial Automation Systems - Product Data Representation and Exchange - Part 42: Integrated Generic Resource: Geometric and Topological Representation.* ISO 10303-42:2003, International Organization for Standardization.

ISO (2020). *Building Information Modelling (BIM) - Data templates for construction objects used in the life cycle of built assets - Concepts and principles.* ISO 23387:2020, International Organization for Standardization.

JORDAHL GmbH (2020). Technische Information: JORDAHL® Schienen und Zubehör. Viewed 2021-06-28, https://jordahl-group.com/fileadmin/user_upload/jordahl-group.com/downloads/Broschueren/de/JOR_LIT_BR_SUZ_DE.pdf.

Lee, G., Sacks, R. & Eastman, C. M. (2006). Specifying Parametric Building Object Behavior (BOB) for a Building Information Modeling System. *Automation in Construction, Knowledge Enabled Information System Applications in Construction.* 15 (6). pp. 758-776. DOI:10.1016/j.autcon.2005.09.009

Mäntylä, M. (1988). *Introduction to Solid Modeling*. W. H. Freeman & Co. New York, NY, USA.

Object Management Group (2017). *Unified Modeling Language (UML) Specification, Version 2.5.1*. https://www.omg.org/spec/UML/2.5.1/ (Visited on 2021-04-13)

Singh, M. M., Anil, S. & Borrmann, A. (2015). Modular Coordination and BIM: Development of Rule Based Smart Building Components. *Proc. of Creative Construction Conference 2015*, Krakow, Poland, June 21st – 24th 2015. pp. 519-527. DOI:10.1016/j.proeng.2015.10.104

Shah, J. J. & Mäntylä, M. (1995). *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. John Wiley & Sons, Inc. 605 Third Ave. New York, NY, USA.

Solihin, W. & Eastman, C. (2015). Classification of Rules for Automated BIM Rule Checking Development. *Automation in Construction* 53 (May 2015). pp. 69–82. DOI:10.1016/j.autcon.2015.03.003.

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 (September 2020), 103248. DOI:10.1016/j.autcon.2020.103248

Stroud, I. (2006). *Boundary Representation Modelling Techniques*. Springer London.

Verein Deutscher Ingenieure (2009). *3D product modelling - Technical and organizational requirements, Procedures, tools, and applications, Cost-effective practical use.* VDI 2209:2009, Verein Deutscher Ingenieure.

Zhang, C., Beetz, J. & de Vries, B. (2018). BimSPARQL: Domain-Specific Functional SPARQL Extensions for Querying RDF Building Data. *Semantic Web*. 9 (6). pp. 829–55. DOI:10.3233/SW-180297

Zhang, S., Teizer, J., Lee, J.-K., Eastman, C. & Venugopal, M. (2013). Building Information Modeling (BIM) and Safety: Automatic Safety Checking of Construction Models and Schedules. *Automation in Construction*. 29 (January). pp. 183–95. DOI:10.1016/j.autcon.2012.05.006