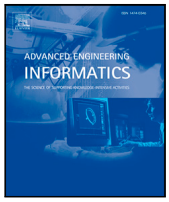
Advanced Engineering Informatics 60 (2024) 102443

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei



Full length article

Comparative analysis of approaches for automated compliance checking of construction data

Emma Nuyts [a,*], Mathias Bonduel [b], Ruben Verstraeten [a]

[a] Department of Architecture and Urban Planning, Ghent University, Jozef Plateaustraat 22, Ghent, 9000, Belgium
[b] Neanex Technologies, Klokstraat 12, Antwerp, 2600, Belgium

ARTICLE INFO

Dataset link: OSF, https://osf.io/5rwt6/?view_only=7637d2f2718341a59a5bd6635965e973

Keywords:
Automated compliance checking
Building information management
Comparative analysis

ABSTRACT

While the domain of Automated Compliance Checking (ACC) has gained track, the construction industry has been flooded with different approaches. This paper studies these different approaches for use in compliance checking of construction data. The approaches are compared by defining constraints for the same set of five requirements, each of a different category, stemming from the Flemish building regulation on accessibility. Eight approaches have been selected for comparison: two IFC-based approaches (Solibri Model Checker and the upcoming buildingSMART standard IDS), two general data standards and their accompanying schema definition languages (JSON Schema and XSD), and four Linked Data approaches (OWL, SWRL, SPARQL, and SHACL). Besides the pure functional analysis, the relative uptake and support in tooling are also considered. While XML/XSD and JSON/JSON Schema and the Linked Data approaches are in essence domain-independent, only the latter has an extra layer for agreeing on high-level data modeling (and thus data validation) patterns in the construction domain with the EN17632-1:2022 standard. SHACL is considered the most adept method from the Linked Data approaches since it is fully standardized for both inputs and outputs and was developed for validation use cases.

1. Introduction

The Architecture, Engineering, Construction, and Operations (AECO) industry has seen a rapid improvement in digitization in the last decade. The design of the built environment has evolved from 2D line drawings on paper and unconnected datasets to using data-centered construction processes relying on databases and 3D modeling, including Building Information Management (BIM) tools combining geometry and alphanumeric data. Examples of alphanumeric data are non-physical objects or semantic information about construction components. While this construction information is captured in machine-readable data, the building regulations, however, are still written in a solely human-readable format. This manual interpretation of the governing codes and standards is known to be error-prone, inefficient, and therefore a contributing factor to declining productivity in the domain [1]. The automation of this compliance checking process would be a vast improvement for the industry, although it is not that easy to implement. The use of Automated Compliance Checking (ACC) would mean both the normative data, stemming from the building regulation, and the construction information have to be represented in a compatible machine-readable format. Fig. 1 shows that the subject from construction project data (stemming from architectural design, energy analysis, circularity calculations, planning tools, etc.) needs to be matched to the corresponding requirement,[1] to allow for verification of this subject. The optimal approach for representing these requirements in a machine-readable format is dependent on the representation of the construction information, to ensure an efficient matching process. Since there are dozens of approaches to define both normative and construction information for use in compliance checking, this article will review existing approaches and compare them. The paper is organized as follows. In Section 2, the processing of normative data, previously studied compliance checking methods and similar comparative analyses are reviewed. In Section 3, the research methodology is proposed, after which eight approaches for compliance checking are evaluated and compared in Section 4. The discussion is presented in Section 5 and final conclusions are given in Section 6.

* Corresponding author.
E-mail addresses: emma.nuyts@ugent.be (E. Nuyts), mathias.bonduel@neanex.com (M. Bonduel), ruben.verstraeten@ugent.be (R. Verstraeten).

[1] The terms ‘requirement’, ‘constraint’, and ‘rule’ will be used interchangeably in this paper.

https://doi.org/10.1016/j.aei.2024.102443
Received 31 August 2023; Received in revised form 17 February 2024; Accepted 22 February 2024

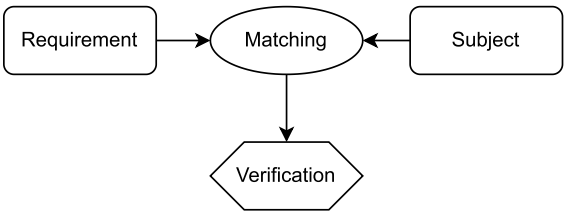


Fig. 1. Compliance checking model.

2. Related work

2.1. Processing normative data

The first step in the creation of an ACC workflow is the processing of the legislative text to create machine-readable constraints. This area of research is also known as Rules as Code (RaC), which is defined as "the activity of creating or converting legal text which is in a natural language, in or into a representation in a computer-processable form" [2]. Doing this conversion manually is not recommended, since experience from previous research indicates it takes an expert a day to translate one page of regulations, including quality control [1]. This conversion can however partially be automated, by using Natural Language Processing (NLP) techniques. There is plenty of research to be found on using NLP for the creation of machine-readable constraints, based on (building) regulations. Some examples of NLP research focusing specifically on use in ACC systems are Zhang & El-Gohary [3], who extract regulatory information from legislative documents and align the representation with building information so that they can be interpreted together in one system. Zhou et al. [4] propose a novel automated rule interpretation framework, based on NLP and context-free grammar. This method achieves 99.6% parsing accuracy for simple sentences and 91.0% for complex sentences.

2.2. Previously studied compliance checking methods

2.2.1. IFC-based approaches

The Industry Foundation Classes (IFC) [5] is a standardized open and neutral data interchange format, specifically intended for the description of AECO models. Different IFC-based approaches for compliance checking exist, ranging from software tools to buildingSMART standards, such as Model View Definitions (MVD) and the upcoming Information Delivery Specifications (IDS) and research initiatives such as the Query Language for Building Information Models (QL4BIM) [6] and BIMRL [7].

ACC tools operating on datasets stored in the open IFC format include (but are not limited to) DesignCheck [8], ePlanCheck and Fornax [9,10], Solibri Model Checker [11], Jotne EDM and SMARTcodes [12] as listed by Amor & Dimyadi [1]. Out of this list, Solibri Model Checker (SMC) is the only one still available and widely used in the industry. It provides a user interface for compliance checking, although the created requirements are not available outside of the software. Apart from software tools, buildingSMART standards such as MVD and IDS can be used to validate data captured in IFC. MVD are defined as a specific implementation level of IFC to describe or facilitate a specific use or workflow [13]. MVD for IFC are created using mvdXML, a buildingSMART standard, of which the latest version (1.1) was published in 2016 [14]. The use of MVD in compliance checking has been researched by Lee et al. [15], who investigated how to create different kinds of constraints based on the IfcDoc tool [16]. However, in February 2023, buildingSMART announced that MVD will be replaced by the future IDS standard for end-users, while the creation of MVD will be left to data developers [17]. Since this standard is still under development, it has not been researched extensively for compliance-checking purposes, except for checking information availability [18].

As for research initiatives, QL4BIM has been proposed by Daum & Borrmann in 2014 [6] as a query language that provides metric, directional, and topological operators for defining filter expressions with qualitative spatial semantics. This query language was extended by Preidel et al. [19], by creating a visual variant of QL4BIM for general IFC filtering and processing. On top, they developed the Visual Code Checking Language (VCCL), suitable for translating building code contents in a digital format. Another research initiative, BIMRL, was proposed by Solihin et al. [7] in 2017. It is based on the BIMRL simplified schema, which allows almost lossless IFC data transformation and utilizing the established relational database system, and supporting highly efficient queries to be performed [7]. Dimyadi et al. [20] integrate BIMRL into a computer-aided compliance audit process to facilitate information gathering.

2.2.2. General open data standards

Apart from conventional BIM datasets, general data formats such as Comma-Separated Values (CSV), the Extensible Markup Language (XML), and the JavaScript Object Notation (JSON) are often used by tool developers in data exchanges. CSV is a structured data format, utilizing commas to separate individual data fields. While RFC 4180 [21] proposes a specification for CSV, it is not standardized. Some CSV schema languages exist [22–24]. The main limitation of using CSV in the AECO industry is that all project data has to fit one table, making it difficult to define relationships between objects. Secondly, XML is a W3C standardized markup language and data format for storing, transmitting, and reconstructing data [25,26]. Multiple schema languages for XML exist, with the XML Schema Definition (XSD) being one of the most common ones. XSD provides a detailed description of a particular type of XML document, specifying additional constraints on the structure and content beyond what is required by the XML syntax. It is furthermore a W3C recommendation [27]. Lastly, JSON is a data format for storing and transmitting data objects consisting of attribute–value pairs and arrays. It was standardized in 2013 and last updated in 2017 [28]. The JSON Schema language includes constraints and conditions, such as whether an information field is mandatory or not, and what datatype a data field should be. A schema created with JSON Schema can be used to check whether the captured data complies with it [29].

2.2.3. Linked data approaches

The concept of Linked Data makes use of the Resource Description Framework (RDF) data model, which is a framework for representing information on the Web and can be stored in one of its many serializations (RDF/XML, JSON-LD, Turtle, etc.) [30]. Linked Data approaches such as the Web Ontology Language (OWL), the Semantic Web Rule Language (SWRL), N3Logic, the SPARQL Protocol and RDF Query Language (SPARQL), and the Shapes Constraint Language (SHACL) can be used for compliance checking of construction data captured as RDF graphs. The W3C standardized Web Ontology Language (OWL) [31] can be used to describe concepts in an ontology, which in turn can be published as an RDF graph to enhance reuse and extensions in other ontologies. While standard OWL axioms can be used in an inferencing process to derive statements, Tao et al. [32] defined a separate variant of OWL that can define so-called "integrity constraints" and operate under different assumptions to cater to a limited amount of actual data validation use cases. Based on OWL and RuleML, the Semantic Web Rule Language (SWRL) [33] was proposed as a W3C member submission in 2004. SWRL defines rules in the form of an implication between an antecedent and a consequent. Uhm et al. [34] created computer-interpretable rules in SWRL, based on the South Korean online e-procurement system. In 2007, N3Logic [35] was proposed as a minimal extension of RDF, ensuring the same language can be used for both logic and data. While it was never accepted as a standard, Pauwels et al. [36] implemented N3Logic for checking acoustical performance.

Another W3C recommendation is SPARQL [37], which is a query language for RDF. Zhong et al. [38] evaluated this language for compliance checking purposes, and Zheng et al. [39] developed an algorithm to automatically generate SPARQL-based queries from regulatory texts. Lastly, SHACL [40] is a language for validating RDF graphs against a set of conditions and a W3C recommendation since 2017. Soman et al. [41] evaluated this language for scheduling constraints in a look-ahead planning environment. Another language for validating RDF graphs is Shape Expressions (ShEx) [42]. However, it is not standardized and has not been used in literature on compliance checking of construction data.

2.3. Similar comparative analyses

Only a few similar comparative analyses have been conducted in the past. Pauwels & Zhang [43] analyze three different strategies: (1) "hard-coded rule checking after querying for information", (2) "rule-checking by querying", and (3) "semantic rule-checking with dedicated rule languages". The first strategy relies on RDF and OWL, while the second strategy uses SPARQL to query the dataset. For the third strategy, both SWRL and N3Logic are provided as options, but the exemplary listing shows the N3Logic notation.

The conclusions state that strategy 1 has a short implementation time, however, customization is limited and knowledge inference is not possible. Strategies 2 and 3 both have a longer implementation time but do allow customization. The second strategy however only supports limited knowledge inference, while the latter fully supports it. A more recent conference paper comparing compliance checking methods was written by authors affiliated with buildingSMART [18]. The paper investigates nine methods (spreadsheet, Product Data Templates, data dictionary, IDS, mvdXML, idmXML, Level Of Information Need, IFC Property Templates, and SHACL) to define information availability requirements, which are evaluated against 19 criteria, divided into 7 categories: (1) Standardized, (2) Applicability, (3) Fields: information type, datatype, unit of measurement, description, references, (4) Value constraints: equality, range, enumeration, patterns, (5) Content: existence, documents, structure, (6) Geometry: representation, detailedness and (7) Metadata: purpose, actors, process map. The authors conclude that none of the researched solutions cover all evaluation criteria, but IDS is presented as "the most advantageous method when it comes to automated compliance checking by validation of the alphanumerical IR". However, no supporting listings were shown.

3. Research methodology

3.1. Comparative analysis

Since there are dozens of compliance checking approaches, this article makes a comparative analysis between different approaches. The primary objective is to define the strengths and weaknesses of each approach, to facilitate a comparison. In Section 3.2, the selection of approaches to be reviewed will be substantiated. To ensure the comparison happens in a consistent manner, the same set of constraints is defined manually with each approach, which is elaborated in Section 3.3. Section 3.4 will cover that although the construction data is needed in multiple formats, attention has been given to the consistency of these formats to ensure standardization. Furthermore, to ensure full reproducibility of the analysis, both the construction project data and the machine-readable constraints for each approach are available on OSF.[2]

3.2. Compliance checking methods reviewed

For the proprietary tools, only Solibri Model Checker will be evaluated, since this is the only one that is still available, is used in the AECO industry, and supports advanced constraints for compliance checking. As for the IFC-based approaches, the upcoming buildingSMART IDS standard will be evaluated. The MVD standard will not be evaluated, due to its replacement by IDS for end-users. QL4BIM and BIMRL will not be investigated further, since they have not reached the level of standardization of aforementioned approaches. For the general open data standards, the XSD schema and JSON Schema will be evaluated, while CSV is skipped due to its general limitation concerning the definition of relationships. Lastly, all previously mentioned W3C Standards regarding Linked Data approaches will be investigated, being OWL, SWRL, SPARQL, and SHACL. This means N3Logic and ShEx are considered out of scope for this paper.

The main differences with the comparative analyses discussed in Section 2.3, are that this paper not only focuses on conventional BIM datasets but also evaluates general data standards (XSD and JSON Schema), provides all necessary pieces to reproduce each case (datasets, machine-readable constraints and concrete steps to reproduce the checking programmatically), and takes the validation output quality of the approaches into account.

Table 1
Categories of requirements extracted from the Flemish building regulation on accessibility (translated to English).

Category	Constraint example
1. information availability	Each door should have exactly one door width property.
2. value	Art. 22§2: The structural dimensions of entrances or doorways must be at least 105 cm wide so that after finishing, a free and level passage width of at least 90 cm is guaranteed.
3. relational	Art. 20§4: A railing must be provided on both sides of the staircase.
4. mathematical	Art. 20§3: The sum of two times the riser and once the tread of each step should be between 57 and 63 cm or a multiple thereof.
5. conditional	Art. 19§1: The slope is at most 6.25 percent for level differences between 25 cm and 50 cm.

3.3. Interpretation of regulation and categories of constraints

From a conceptual point of view, we identify five categories of requirements to be tested: information availability, value, relational, mathematical, and conditional constraints. To get a thorough sense of the strengths and weaknesses of the different approaches, five requirements (one per category) were distilled from the Flemish regulation on accessibility [44] as shown in Table 1. The interpretation of the textual requirements to form machine-readable constraints will be done manually by the authors in this analysis since a full automation of the extraction of machine-readable requirements from building legislation is out of scope for this paper. The first check should be to evaluate if the required information is available in the construction data, to ensure that a component is not skipped in the checking procedure if the required property is not present. Secondly, value constraints allow to check whether a value is more/less than or equal to a nominal value as defined in the building regulations. Furthermore, relational constraints (category 3) are a means of checking whether a component is present in relation to another component. Moreover, building codes often constrain the result of a mathematical formula that can be calculated using several properties defined in the project data (category 4). If the result of the formula is already present in the construction data, the constraint is a value constraint (category 2) rather than a mathematical constraint. Lastly, conditional constraints (category 5) are used to define multiple cases of boundary values of one property. The legislation is often written in an 'if…then…' manner in such cases.

However, not every article of the building regulation can be represented by one of these categories of requirements, which leads to a sixth

[2] https://osf.io/5rwt6/?view_only=7637d2f2718341a59a5bd6635965e973

category: expert intervention. This category covers e.g. performance-based regulations, or complex energy analyses, and will not be evaluated in this paper. Furthermore, direct validation of the geometries in construction datasets is also out of the scope of this paper. Alphanumeric properties can be derived from geometry using a separate dedicated process. Consequently, a wider variety of commonly used compliance checking methods can be considered. The compliance checking tools do not need to be aware of the original geometry descriptions that might exist in a wide variety of formats and geometry representations (meshes, BREP, NURBS, etc.) [45].

3.4. Construction project data preparation

As an example of project data, a simple three-story, 18-room building model has been created in Autodesk Revit 2023. We created four variants of the building model, to correctly evaluate different compliance checking methods and realistic data scenarios. To start, we created one Revit dataset that complies with all example requirements and one that violates each example requirement, both using different units for numeric properties compared to the regulations. Both variants ("pass" and "fail") are also included in a simulated preprocessing step to align the units of the dataset to the ones of the requirements (resp. "pass_preprocessed" and "fail_preprocessed"). Since no constraint-executing mechanisms are tested in the authoring tool as such, we export the four variants of the project dataset using the IFC standard. An alternative method builds upon general Linked Data standards, where domain-specific ontologies are introduced, conceptualizing the project data as interrelating objects with properties and references to the respective geometrical models. We wanted to capture the same content in a simple yet rich structure following the EN 17632-1:2022 [46] standard, but since no such translation tool is currently available,[3] we manually translated the relevant content from the IFC files to RDF graphs. The result is a set of small but well-structured example datasets (one for each original variant) which suffices for our further analysis. The content of the same manually generated RDF graph is used to generate example XML and JSON datasets using an arbitrary XML and JSON structure that can represent the relevant project content. Instead of translating the full IFC-STEP file into an IFC-XML or IFC-JSON serialization, we end up with concise, easy-to-understand, and well-structured XML and JSON example datasets for our evaluation purposes. In all examples used in this paper, two main assumptions are made: the construction components are classified correctly, and the names of the classifications, properties, and units are aligned between the project data and the constraints.

4. Requirement execution

For the requirement execution, the selected approaches will be defined for the five constraints, allowing a standardized comparison of the methods, as shown in Fig. 2. The listings shown in this Section will only be exemplary snippets, but the full datasets (construction project data, constraints, and if applicable validation output) can be found at OSF.[4] To keep the examples concise, units are not explicitly checked, but this can be done using a similar approach to the information availability constraint. The units should however be checked in a future implementation, to minimize the chance of incorrect results (false positives or false negatives). The prefixes used in the examples are listed in Listing 1, in which self-defined prefixes are denoted with an asterisk.

[3] The existing converters that can turn IFC-STEP datasets in RDF graphs structure the graph using the complex ifcOWL ontology (e.g., the IFC-to-RDF converter [47]) or the more concise but less complete and not standardized Linked construction data ontologies BOT and BEO/MEP/FURN (e.g., the IFCtoLBD converter [48]). In the latter case, for instance, units are not included and URIs for properties are generated on the fly without being defined in any ontology.

[4] https://osf.io/5rwt6/?view_only=7637d2f2718341a59a5bd6635965e973

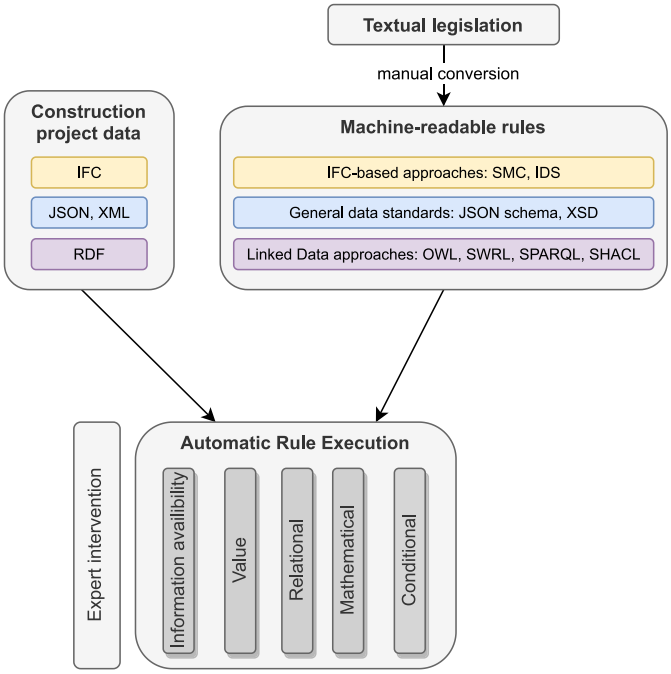


Fig. 2. Comparative analysis methodology.

Listing 1: Prefixes used

```
ids    <http://standards.buildingsmart.org/IDS>
ifcotl* <https://otl.buildingsmart.org/IFC4_ADD2_TC1/def/>
otl*   <http://otl.company-x.com/def/>
owl    <http://www.w3.org/2002/07/owl#>
rdf    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
rdfs   <http://www.w3.org/2000/01/rdf-schema#>
reg*   <https://accessibility-regulations.org/shapes/>
sh    <http://www.w3.org/ns/shacl#>
sml    <https://w3id.org/sml/def#>
swrlb   <http://www.w3.org/2003/11/swrl#builtin>
xs    <http://www.w3.org/2001/XMLSchema>
xsd    <http://www.w3.org/2001/XMLSchema#>
```

4.1. Requirement execution with approaches on IFC data

4.1.1. Solibri Model Checker (SMC)

Solibri Model Checker [11] is a proprietary BIM application that operates on IFC, where construction models from different BIM authoring tools can be imported into an IFC interface. The compliance checking is based on both pre-defined and customizable rules within hard-coded boundaries using a graphical user interface. The rules are however not formalized in a standard manner and not available outside of the application. The ruleset manager furthermore allows user input such as the type of building, to complete the construction information stored in the BIM model. This application differs in workflow implementation from other approaches reviewed, since it allows for communicating about issues through the BIM Collaboration Format (BCF) directly in the interface. Furthermore, the checking process can be automated using Solibri Autorun, ensuring that compliance is checked regularly during the design process. The screen captures in this Section are from Solibri Anywhere version 9.13.3.18.

The first type of constraint, the information availability constraint, can be created using Property Sets, as shown in Fig. 3. This property set defines that each entity of a component, doors in this example, must have a specified property, such as OverallWidth. By using the asterisk wildcard in the value condition, the constraint only checks if the property exists and has a value.

Fig. 3. Information availability constraint using SMC.

Fig. 4. Value constraint using SMC.

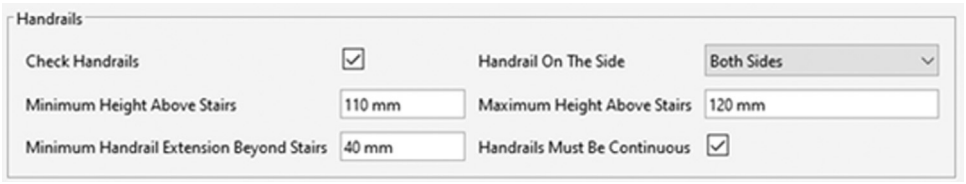


Fig. 5. Relational constraint using SMC.

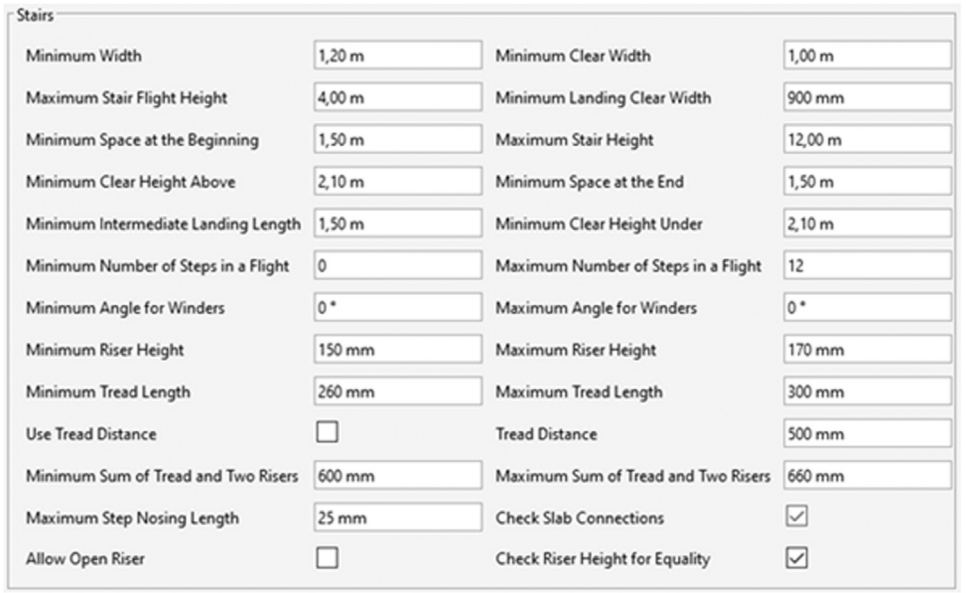


Fig. 6. Mathematical constraint using SMC.

The value constraint shown in Fig. 4 can be defined similarly to the information availability constraint, except a value is specified instead of the wildcard of the previous example.

Relational constraints, for example defining that a stair should have a hand-rail on both sides, can also be checked. Fig. 5 shows that the positioning of handrails on the sides can be set to "Both Sides". Next, geometrical constraints on the handrail can be checked, such as the minimum and maximum height or the extension beyond the stairs.

Furthermore, mathematical constraints can be defined, such as setting the minimal and maximal values of the sum of a tread and two risers of a staircase, as shown in Fig. 6. As with the handrail, Solibri includes built-in algorithms to derive alphanumeric data from the geometric representation.

Lastly, conditional constraints can be defined by providing the values in the table shown in Fig. 7. Ramps can be checked by defining a slope and a length, or with a rise and a gradient, depending on the structure of the written building legislation. Furthermore, the minimal space at the beginning of the ramp can be checked, which is a benefit for assessing the turning circles of wheelchairs for example.

The Solibri Model Checker can define the five categories of constraints used for the comparison, in a user-friendly interface. The rules included in the ruleset manager are already quite extensive, and these just had to be customized with the correct values for this implementation. However, a full automation of the workflow to go from human-readable building legislation to Solibri rules will not be straightforward, since the rules are not available outside of the application and it is not disclosed how they should be defined for bulk import.

4.1.2. Information Delivery Specification (IDS)

The Information Delivery Specification (IDS) [49] is a buildingSMART standard under development primarily targeted at checking information requirements from IFC models. An IDS file contains a list of specifications in XML syntax, which can be used to evaluate IFC models. IDS uses facets, which describe information that a single

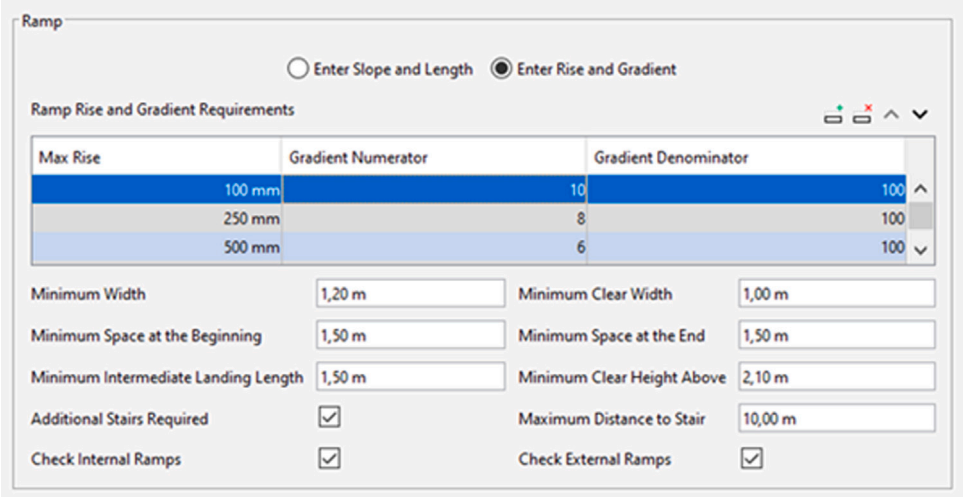


Fig. 7. Conditional constraint using SMC.

entity (e.g. wall, door, etc.) in the model may have. Different kinds of facets can be created using IDS, such as entity facets, attribute facets, classification facets, property facets, material facets, and partof facets. Since the standard is still under development, only a few IDS validators were available at the time of writing. The examples in this paper are created and validated using usBIM.IDS version 2.4.68 of ACCA software [50]. The construction data represented in IFC and the IDS constraints can be found at OSF:IDS.[5]

The information availability constraint can be created using either attribute or property facets. Attributes are defined as a limited set of fundamental data for an IFC model, while properties are defined as additional data that describes an IFC element. Whether an attribute or property facet is needed, thus depends on the type of IFC data that needs to be validated. Listing 2 shows how the attribute facet for the width of a door is created. The applicability is an IfcDoor, which is required to have an attribute named OverallWidth. This attribute can only occur exactly once, which is defined by setting both `minOccurs` and `maxOccurs` to one.

Listing 2: Information availability constraint using IDS

```
<ids:applicability>
  <ids:entity>
    <ids:name>
      <ids:simpleValue>IfcDoor</ids:simpleValue>
    </ids:name>
  </ids:entity>
</ids:applicability>
<ids:requirements>
  <ids:attribute minOccurs="1" maxOccurs="1">
    <ids:name>
      <ids:simpleValue>OverallWidth</ids:simpleValue>
    </ids:name>
  </ids:attribute>
</ids:requirements>
```

The same pattern of attribute and property facets can be extended to evaluate the value constraint, as shown in Listing 3. In addition to the previous constraint, the constraint now contains a value restriction, defining that the minimal OverallWidth of the IfcDoor should be 900 (assuming millimetres). This is done with the `xs:minInclusive` construct. For other value constraints, `xs:maxInclusive`, `xs:minExclusive`, and `xs:maxExclusive` can be used.

[5] https://osf.io/jkh2z/?view_only=f8c94f79e0e14c5e8134486fd81b0e62

Listing 3: Value constraint using IDS

```
<ids:applicability>
  <ids:entity>
    <ids:name>
      <ids:simpleValue>IfcDoor</ids:simpleValue>
    </ids:name>
  </ids:entity>
</ids:applicability>
<ids:requirements>
  <ids:attribute minOccurs="1" maxOccurs="1">
    <ids:name>
      <ids:simpleValue>OverallWidth</ids:simpleValue>
    </ids:name>
    <ids:value>
      <xs:restriction base="xs:decimal">
        <xs:minInclusive value="900" />
      </xs:restriction>
    </ids:value>
  </ids:attribute>
</ids:requirements>
```

Lastly, relational constraints can be defined using a partof facet, in which IDS supports four types of relationships: IfcRelAggregates, IfcRelAssignsToGroup, IfcRelContainedInSpatialStructure, and IfcRelNests. The constraint in Listing 4 ensures that each stair contains two railings, by using the IfcRelAggregates relation in combination with setting both `minOccurs` and `maxOccurs` equal to two. However, this kind of constraint cannot check the relative placement of the railing, meaning the assumption is made that two railings cannot be placed on the same side of the staircase.

Listing 4: Relational constraint using IDS

```
<ids:applicability>
  <ids:entity>
    <ids:name>
      <ids:simpleValue>IfcStair</ids:simpleValue>
    </ids:name>
  </ids:entity>
</ids:applicability>
<ids:requirements>
  <ids:partOf relation="IfcRelAggregates" minOccurs="2"
    maxOccurs="unbounded">
    <ids:entity>
      <ids:name>
        <ids:simpleValue>IfcRailing</ids:simpleValue>
      </ids:name>
    </ids:entity>
  </ids:partOf>
</ids:requirements>
```

Since the first version of IDS targets basic information and relationships in IFC, more advanced information requirements such as mathematical and conditional constraints are currently out of scope [51]. Since this standard is currently under development and software implementations are scarce at the time of writing, the ease of implementation in architectural workflows will have to be investigated further. However, the XML syntax allows for automation from written building legislation to machine-readable constraints.

4.2. Requirement execution with general open data standards

4.2.1. XML Schema Definition Language (XSD)

The XML Schema Definition Language (XSD) [25] is standardized by the W3C and used to define the structure, datatypes, and constraints of XML documents. It provides a standardized way to describe the allowed elements, attributes, and their relationships within an XML document. XSD is useful for defining the structure of XML documents, i.e. defining an XML schema for a certain purpose. It focuses on describing the devised XML format and validation of XML data against the schema rather than capturing the semantics or meaning of the data. XSD provides a way to specify constraints such as required elements, datatypes, and cardinalities of relations and properties, but it does not support reasoning or logical axioms. Since XSD is used in many more domains than the AECO industry, multiple implementations can be found. An online XSD validator [52] based on the Apache Xerces-J library [53] was used to validate the five exemplary constraints in this paper, however, any validator that supports XSD 1.1 can be used. The construction data represented in XML and the XSD constraints can be found at OSF:XSD.[6]

The information availability constraint is defined by stating that each door component should have a property OverallWidth. This property should occur exactly once, which is defined by setting both `minOccurs` and `maxOccurs` to one, as shown in Listing 5. This example clearly shows XSD was developed with this purpose in mind since the constraint is defined with a relatively straightforward construct.

Listing 5: Information availability constraint using XSD

```
<xs:element name="door" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OverallWidth" type="xs:double"
     minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

To define a value constraint on the door width, a similar approach is used. However, as shown in Listing 6, the OverallWidth property now has an additional restriction, stating with `xs:minInclusive` that the minimal value should be 900 (assuming millimetres). In parallel with IDS, similar constraints can be defined using `xs:maxInclusive`, `xs:minExclusive`, and `xs:maxExclusive`.

Listing 6: Value constraint using XSD

```
<xs:element name="door" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OverallWidth">
        <xs:simpleType>
          <xs:restriction base="xs:double">
            <xs:minInclusive value="900" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

[6] https://osf.io/3dxag/?view_only=077f44c956c54e1f82b8a6ed13ba4bda

For the relational constraint, the structure of the XML dataset is important. In the prepared XML project dataset, the railing is nested in the stair component, leading to a similar construct as the information availability constraint. However, in this case, there should be two railing components present, as shown in Listing 7, which is defined by setting both `minOccurs` and `maxOccurs` equal to two. Once more, the constraint is simplified to evaluate the presence of two railings, and their relative placement on opposite sides of the staircase is not checked.

Listing 7: Relational constraint using XSD

```
<xs:element name="stair" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="railing" minOccurs="2"
     maxOccurs="unbounded" />
      <xs:element name="riserHeight" type="xs:double" />
      <xs:element name="treadLength" type="xs:double" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

For mathematical (and conditional) constraints, the `xs:assert` tag is needed. This XSD 1.1 technique allows for defining a test that the defined properties should fulfill. Listing 8 shows the test defining that the result of two times the riser height plus the tread length should be between 570 and 630. It is important to note that some characters in the test should be encoded, such as encoding < as `<` to ensure a correct validation.

Listing 8: Mathematical constraint using XSD

```
<xs:element name="stair" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="railing" minOccurs="2"
     maxOccurs="2" />
      <xs:element name="riserHeight" type="xs:double" />
      <xs:element name="treadLength" type="xs:double" />
    </xs:sequence>
    <xs:assert test="(riserHeight * 2 + treadLength) >=
     570 and (riserHeight * 2 + treadLength) &lt;= 630" />
  </xs:complexType>
</xs:element>
```

Conditional constraints also make use of the test in the `xs:assert` tag, however now an if-then statement is needed. Listing 9 shows the slope should be less than 6.25% if the height difference lies between 250 and 500. In this example, it is once more important to encode particular characters, and to let a non-inclusive if-then statement end with 'else false()'.

Listing 9: Conditional constraint using XSD

```
<xs:element name="ramp" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="heightDifference" type="xs:double"
     />
      <xs:element name="slope" type="xs:double" />
    </xs:sequence>
    <xs:assert test="if (heightDifference > 250 and
     heightDifference &lt; 500) then slope &lt; 0.0625 else
     false() "/>
  </xs:complexType>
</xs:element>
```

The XSD schema was able to define all five categories of constraints. Furthermore, since this standard was developed almost two decades ago and has been integrated in many domains since then, the documentation and examples to be found online are quite extensive.

4.2.2. JavaScript object notation (JSON) schema

JSON Schema [29] is a language to describe the structure of JSON data. It provides a standardized way to define requirements and validations for JSON documents, allowing for compliance checking against those requirements. It supports defining a set of constraints that the JSON data must adhere to, such as datatypes, required fields, allowed values, and more complex validations. To validate the constraints against a given set of requirements, a JSON Schema validator should be used. Since this standard has been around for quite some time and is adopted in many domains, multiple validators can be found, just like with the XSD schema. The examples in this paper are validated using the JavaScript Ajv library [54]. The construction data represented in JSON and the JSON Schema constraints can be found at OSF:JSONSchema.[7]

An information availability constraint is defined by stating that a property of a component is required. As Listing 10 shows, a door component has a property OverallWidth, which is a number. The property is then declared as required for each door.

Listing 10: Information availability constraint using JSON Schema

```
"door": {
 "type": "object",
 "properties": {
  "OverallWidth": {
   "type": "number"
  }
 },
 "required": ["OverallWidth"]
}
```

A value constraint is specified similarly to the information availability constraint, however, the OverallWidth property now gets a minimal value of 900 assigned, as shown in Listing 11. The `minimum` keyword is defined as being less or equal to the specified number. For similar constraints, `maximum`, `exclusiveMinimum` and `exclusiveMaximum` can be used.

Listing 11: Value constraint using JSON Schema

```
"entrance": {
 "type": "object",
 "properties": {
  "OverallWidth": {
   "type": "number",
   "minimum": 900
  }
 }
}
```

Similar to the XSD schema, the definition of relational constraints is dependent on the used JSON structure. In the exemplary JSON file, the railings are nested in the staircase component. This structure leads to the JSON Schema definition in Listing 12, which defines that each staircase should have an array of railings, with a minimal length of two. The nesting of the railings inside the staircase object is compulsory, since JSON Schema does not allow for referring to other objects. Once again, the relative placement of the staircase and the railings is not explicitly checked.

Listing 12: Relational constraint using JSON Schema

```
"staircase": {
 "type": "object",
 "properties": {
  "railing": {
   "type": "array",
   "minItems": 2
  }
 }
}
```

The JSON Schema does not allow for mathematical operators to be defined, meaning the fourth type of constraint cannot be defined with this approach. Conditional constraints however are possible, by using an if-then statement. The schema in Listing 13 defines that if the height difference property of the ramp has a minimal value of 250 and a maximal value of 500, the slope should have a maximal value of 6.25%.

Listing 13: Conditional constraint using JSON Schema

```
"ramp": {
 "type": "object",
 "if": {
  "properties": {
   "heightDifference": {
    "type": "number",
    "minimum": 250,
    "maximum": 500
   }
  }
 },
 "then": {
  "properties": {
   "slope": {
    "type": "number",
    "maximum": 0.0625
   }
  }
 }
}
```

JSON Schema provides a standardized way to express only four categories of constraints since the mathematical calculation could not be defined. The schema is however fully defined by the structure of the construction data, which is not standardized in the AECO industry. This is particularly important to create relational constraints since there are no methods to reference other objects. Furthermore, the JSON Schema specification describes a standardized validation output, however, it is not compulsory for JSON Schema validators to implement this.

4.3. Requirement execution with Linked Data approaches

4.3.1. Web Ontology Language (OWL)

The W3C standardized Web Ontology Language (OWL) can be used to describe concepts in an ontology, which in turn can be published as an RDF graph to enhance reuse and extensions in other ontologies. OWL operates under the Open World Assumption (OWA) and No Unique Name Assumption (NUNA). Consequently, it cannot say what is “correct” and “wrong” data in a dataset as all it can do is (1) detect an inconsistency for the dataset and ontology as a whole and (2) derive additional statements if there are no inconsistencies detected. In essence, OWL is not a language for defining actual constraints, but thanks to usage in inferencing processes it can help to simplify query patterns used in actual validation tests, e.g. applying the SPARQL query language. The examples in this paper are validated using Protégé v5.6.1, after which an inferencing process is started using the Pellet reasoner v2.2.0. The construction data represented in LBD and the OWL constraints can be found at OSF:OWL.[8]

The information availability constraint is defined with an OWL statement that flags the individuals that have an attribute OverallWidth, due to the lack of support for NAF rules. Listing 14 shows this is done by using a `owl:Restriction` construct. Next, a SPARQL query is defined to find the objects that violate this requirement, by querying for all individuals of type Door and looking for an absence of the OK flag, shown in Listing 15.

Listing 14: Information availability constraint using OWL

```
[
  owl:intersectionOf
```

[7] https://osf.io/n4389/?view_only=4ed6ece691db42e2b7a6df36198a41d6

[8] https://osf.io/vzw65/?view_only=9f486dea16f34e53a5647eefd3ac00c5

```
  (
    ifcotl:Door
    [
      rdf:type owl:Restriction ;
      owl:onProperty ifcotl:attribute_overallWidth ;
      owl:someValuesFrom [
        rdf:type owl:Restriction ;
        owl:onProperty rdf:value ;
        owl:someValuesFrom xsd:decimal
      ]
    ]
  ) ;
  rdf:type owl:Class ;
  rdfs:subClassOf otl:Req01_ok
] .
```

Listing 15: SPARQL query to retrieve all components minus the ones that are correct

```
SELECT ?failedIndividual
WHERE {
  ?failedIndividual a ifcotl:Door .
  MINUS { ?failedIndividual a otl:Req01_ok . }
}
```

To define a value constraint, an `owl:withRestrictions` construct is used in combination with `xsd:maxExclusive`, as shown in Listing 16. Similar constraints can be created using constructs explained in Section 4.2.1. To retrieve the violating individuals, a SPARQL query as shown in Listing 17 can be used.

Listing 16: Value constraint using OWL

```
[
  owl:intersectionOf
  (
    ifcotl:Door
    [
      rdf:type owl:Restriction ;
      owl:onProperty ifcotl:attribute_overallWidth ;
      owl:someValuesFrom [
        rdf:type owl:Restriction ;
        owl:onProperty rdf:value ;
        owl:someValuesFrom [
          rdf:type rdfs:Datatype ;
          owl:onDatatype xsd:decimal ;
          owl:withRestrictions (
            [ xsd:maxExclusive "900"^^xsd:decimal ]
          )
        ]
      ]
    ]
  ) ;
  rdf:type owl:Class ;
  rdfs:subClassOf otl:Req02_nok
] .
```

Listing 17: SPARQL query to retrieve all components that violate the requirement

```
SELECT ?failedIndividual
WHERE {
  ?failedIndividual a otl:Req02_nok .
}
```

The relational constraint needs a validation process that returns individuals of type Stair that do not have at least two individuals of type Railing connected. This is done by using `owl:minQualified Cardinality` in combination with `owl:onProperty` and `owl:onClass`, as shown in Listing 18. In parallel with the information availability constraint, is this pattern not directly possible with OWL due to the lack of support for NAF axioms. The SPARQL query shown in Listing 15 can be used to retrieve the violating instances. Once more, the relative placement of the railing and the stairs is not explicitly checked.

Listing 18: Relational constraint using OWL

```
[
  owl:intersectionOf
  (
    ifcotl:Stair
    [
      rdf:type owl:Restriction ;
      owl:minQualifiedCardinality
     "2"^^xsd:nonNegativeInteger ;
      owl:onProperty sml:hasPart ;
      owl:onClass ifcotl:Railing
    ]
  ) ;
  rdf:type owl:Class ;
  rdfs:subClassOf otl:Req03_ok
] .
```

Since OWL does not define any mathematical operators, it cannot define the mathematical constraint. Conditional constraints however can be created using OWL, by defining the restriction on the height difference and the slope in an `owl:intersectionOf` construct, as shown in Listing 19. Similar to the value constraint, the SPARQL query of Listing 17 can be used to find the individuals that violate the requirement.

Listing 19: Conditional constraint using OWL

```
[
  owl:intersectionOf
  (
    ifcotl:Ramp
    [
      rdf:type owl:Restriction ;
      owl:onProperty otl:property_heightDifference ;
      owl:someValuesFrom [
        rdf:type owl:Restriction ;
        owl:onProperty rdf:value ;
        owl:someValuesFrom [
          rdf:type rdfs:Datatype ;
          owl:onDatatype xsd:decimal ;
          owl:withRestrictions (
            [ xsd:maxExclusive "500"^^xsd:decimal ]
            [ xsd:minInclusive "250"^^xsd:decimal ]
          )
        ]
      ]
    ]
    [
      rdf:type owl:Restriction ;
      owl:onProperty otl:property_slope ;
      owl:someValuesFrom [
        rdf:type owl:Restriction ;
        owl:onProperty rdf:value ;
        owl:someValuesFrom [
          rdf:type rdfs:Datatype ;
          owl:onDatatype xsd:decimal ;
          owl:withRestrictions (
            [ xsd:minExclusive "0.0625"^^xsd:decimal ]
          )
        ]
      ]
    ]
  ) ;
  rdf:type owl:Class ;
  rdfs:subClassOf otl:Req05_nok
] .
```

The OWL language on its own is not sufficient for a validation process as it is in essence a language for describing concepts with OWL axioms that rely on a set of predefined logical constructs that operate under the Open World Assumption (OWA) and the No Unique Name Assumption (NUNA). A validation process involving OWL still relies on a second step using a SPARQL query engine for retrieving individual objects that violate certain requirements. Nevertheless, OWL can help to reduce the complexity of SPARQL queries thanks to the inferred knowledge. In addition, when using a capable OWL reasoning engine, one can ask the inferencing engine for precise explanations of why a certain inference (e.g. an inferred “flag” for signaling a certain requirement violation or pass) was made. A disadvantage of using OWL in a data validation

process is that it is not straightforward as many people find it counter-intuitive to work with the OWA and NUNA since they expect to be writing closed-world constraints for a set of project data containing disjunct individual objects.

4.3.2. Semantic Web Rule Language (SWRL)

The Semantic Web Rule Language (SWRL) [33] is a language for expressing inferencing rules for reasoning on RDF datasets. The primary documentation of the language, its abstract notation and its binding to the RDF data model is the subject of a W3C member submission document dating back to 2004. This implies that it was never elevated to the level of a formal W3C standard. Since SWRL (in parallel with OWL) works under the No Unique Name Assumption (NUNA), one also needs to be explicit about the similarity and disjointness of things in a dataset, using respectively the `owl:sameAs` or `owl:disjointFrom` axioms. Another challenge are so-called Negation as Failure (NAF) rules. Since these kinds of rules are not supported by SWRL, a workaround is to act on all individuals which are correct instead and assuming during analysis that the other side contains all individuals which are not passing the requirement. Different syntaxes for SWRL exist, such as the abstract syntax, the XML concrete syntax, and the RDF concrete syntax. The examples in this paper will be defined using the concise abstract syntax. The SWRL rules were first created and tested using the SWRLTab5 v2.1.0 [55] in Protégé v5.6.1 [56]. In a follow-up step, the RDF dataset and the SWRL rules were loaded in a fresh Protégé environment to start an inferencing process with the plugin Pellet reasoner v2.2.0. A user can then use SPARQL or the Protégé GUI to retrieve individual objects which violate or respectively pass certain requirements, by querying over the combination of the asserted and inferred statements. The construction data represented in LBD and the SWRL constraints can be found at OSF:SWRL.[9]

The information availability constraint cannot be defined with SRWL due to the lack of support for NAF rules. As a solution, a SWRL rule that flags the correct doors is devised, as shown in Listing 20. The rule flags all doors that have the attribute OverallWidth with a value of OK. Similar to OWL, the SPARQL query in Listing 15 is needed to find the objects that violate this requirement, by querying all individuals of type Door and looking for an absence of the OK flag.

Listing 20: Information availability constraint using SWRL

```
ifcotl:Door(?x) ^ ifcotl:attribute_OverallWidth(?x, ?y) ^
     rdf:value(?y, ?z) -> otl:Req01_ok(?x)
```

The value constraint can be defined using a SWRL built-in (swrlb) for the numeric operator, to infer a flag for individuals that violate the requirement. Listing 21 shows that all doors with an attribute OverallWidth less than 900 are flagged as NOK. A SPARQL query as shown in Listing 17 is needed to retrieve all individuals that violate the value constraint, just like with OWL.

Listing 21: Value constraint using SWRL

```
ifcotl:Door(?x) ^ ifcotl:attribute_OverallWidth(?x, ?y) ^
     rdf:value(?y, ?z) ^ swrlb:lessThan(?z, 900) ->
     otl:Req02_nok(?x)
```

The relational constraint needs a validation process that returns all stairs that are not connected to two railings. In parallel with the information availability constraint, this pattern is not directly possible due to the lack of support for NAF rules. As shown in Listing 22, stairs that are connected to two railings are flagged as OK. Due to the NUNA under which SWRL operates, similar to OWL, the rule explicitly checks that the two individual railings are different, however omitting to check the relative placement of the railings and the stair. The dataset furthermore needs to explicitly state that the two railings are disjoint using `owl:differentFrom`. A similar SPARQL query as in Listing 15 is needed to retrieve the objects that violate the requirement.

[9] https://osf.io/f4j8t/?view_only=7183de5cc177400893037c01b8f0f2fa

Listing 22: Relational constraint using SWRL

```
ifcotl:Stair(?x) ^ sml:hasPart(?x, ?y) ^ sml:hasPart(?x,
     ?z) ^ ifcotl:Railing(?y) ^ ifcotl:Railing(?z) ^
     differentFrom(?y, ?z) -> otl:Req03_ok(?x)
```

Mathematical constraints can be defined in SWRL using built-ins such as multiply, divide, add, and subtract in combination with logical operators such as lessThan, greaterThan, and (not)Equal. As shown in Listing 23, the riser is doubled, after which the sum with the tread is calculated. If the result is less than 570, the stair is flagged as NOK. A similar rule is needed to verify the staircase is not too steep, by using `greaterThan(?sum2, 630)`. In parallel with the value constraint, a simple SPARQL query is needed to find the individual objects that violate the requirement, as shown in Listing 17.

Listing 23: Mathematical constraint using SWRL

```
ifcotl:Stair(?x) ^ ifcotl:property_riserHeight(?x, ?y) ^
     rdf:value(?y, ?z) ^ ifcotl:property_treadLength(?x,
     ?a) ^ rdf:value(?a, ?b) ^ swrlb:multiply(?sum1, ?z, 2)
     ^ swrlb:add(?sum2, ?b, ?sum1) ^ swrlb:lessThan(?sum2,
     570) -> otl:Req04_nok(?x)
```

For more complex constraints such as conditional constraints, it is not possible to define a single SWRL inferencing rule due to the lack of a logical OR construct in SWRL. A simple solution is to define multiple SWRL inferencing rules and execute them together. Listing 24 shows the constraint defining that if the height difference is less than 500 and greater than or equal to 250, and the slope is greater than 6.25%, the ramp should be flagged as NOK. The non-complying components are retrieved using a similar query to the one shown in Listing 17.

Listing 24: Conditional constraint using SWRL

```
ifcotl:Ramp(?x) ^ otl:property_heightDifference(?x, ?y) ^
     rdf:value(?y, ?z) ^ swrlb:lessThan(?z, 500) ^
     swrlb:greaterThanOrEqual(?z, 250) ^
     otl:property_slope(?x, ?a) ^ rdf:value(?a, ?b) ^
     swrlb:greaterThan(?b, 0.0625) -> otl:Req05_nok(?x)
```

While SWRL was able to define the five categories of constraints, it is not sufficient for a data validation process on its own, since it is, in essence, a language for inferencing rules that operates under the Open World Assumption (OWA) and the No Unique Name Assumption (NUNA). A validation process involving SWRL still relies on a second step using a SPARQL query engine for retrieving individual objects that violate or pass certain requirements. Nevertheless, SWRL can help to reduce the complexity of SPARQL queries thanks to the inferred knowledge. In addition, when using a capable SWRL reasoning engine, one can ask the inferencing engine for precise explanations of why a certain inference (e.g. an inferred flag for signaling a certain requirement violation or pass) was made. Writing SWRL inferencing rules with its abstract syntax is actually pretty straightforward thanks to the typical IF-THEN patterns. A drawback, however, is that SWRL is not a W3C standard and only has limited support in existing Linked Data tooling.

4.3.3. SPARQL protocol and RDF query language (SPARQL)

The SPARQL query language for RDF [37] has been the W3C standard for querying RDF datasets since 2013. The language supports four query forms: SELECT, CONSTRUCT, ASK, and DESCRIBE. Since the query language furthermore supports negation, SPARQL queries can be applied to retrieve objects in a dataset that violate certain requirements using SPARQL SELECT. The result of a SPARQL SELECT query is a standardized tabular structure in JSON, XML, or CSV format using the returned values for the query variables. Alternatively, SPARQL CONSTRUCT queries can be used to return an RDF graph, based on the demanded graph pattern. In this research, we work with SPARQL SELECT queries, since they are sufficient for our evaluation.

The construction data represented in LBD and the SPARQL constraints can be found at OSF:SPARQL.[10]

Information availability constraints can be checked easily with a SPARQL query searching for all objects that do not contain certain properties, using `FILTER NOT EXISTS`. This results in a list of all components that violate the given constraints. As shown in Listing 25, doors that do not have an OverallWidth attribute with corresponding value are flagged as NOK and returned as the result.

Listing 25: Information availability constraint using SPARQL

```
SELECT ?invalidObjects ?req
WHERE {
BIND("Req01_nok" AS ?req) .
?invalidObjects a ifcotl:Door .
FILTER NOT EXISTS {
  ?invalidObjects ifcotl:attribute_OverallWidth ?widthProp
     .
  ?widthProp rdf:value ?widthPropVal .
  }
}
```

Similar to the information availability constraint, the value constraint can be checked using a FILTER NOT EXISTS. Listing 26 shows that doors with an OverallWidth attribute of less than 900 are indicated as NOK and returned as the result.

Listing 26: Value constraint using SPARQL

```
SELECT ?invalidObjects ?req
WHERE {
BIND("Req02_nok" AS ?req) .
?invalidObjects a ifcotl:Door .
FILTER NOT EXISTS {
  ?invalidObjects ifcotl:attribute_OverallWidth ?widthProp
     .
  ?widthProp rdf:value ?widthPropVal .
  FILTER(?widthPropVal >= 900)
  }
}
```

Relational constraints can be checked with SPARQL by querying all objects of the `hasPart` aggregate, and defining that two of those objects should be railings with different URIs, as shown in Listing 27. Components that do not correspond to this requirement are indicated as NOK and returned. The relative placement of the railings on both sides of the stair is not explicitly checked, making the assumption that two railings cannot be placed on the same side of the staircase. Since SPARQL operates under a Unique Name Assumption (UNA), it assumes each railing (with a different URI) represents a different railing.

Listing 27: Relational constraint using SPARQL

```
SELECT ?invalidObjects ?req
WHERE {
BIND("Req03_nok" AS ?req) .
?invalidObjects a ifcotl:Stair .
FILTER NOT EXISTS {
  ?invalidObjects sml:hasPart ?railing1, ?railing2 .
  ?railing1 a ifcotl:Railing .
  ?railing2 a ifcotl:Railing .
  FILTER ( ?railing1 != ?railing2 )
  }
}
```

Mathematical constraints can be defined in SPARQL by using normal mathematical operators (*, /, +, -). Logical operators can also be used (=, <, >, >=, <=) to check if a computed value meets the requirements of the constraint. Listing 28 shows that the sum of two times the riser plus one tread is bound as a value, after which the incorrect values are filtered out. Using the same construct as the other constraints, non-complying components are returned by the query.

[10] https://osf.io/yznrx/?view_only=4dda164d79e24132811969271647ac28

Listing 28: Mathematical constraint using SPARQL

```
SELECT ?invalidObjects ?req
WHERE {
 BIND("Req04_nok" AS ?req) .
 ?invalidObjects a ifcotl:Stair .
 FILTER NOT EXISTS {
  ?invalidObjects ifcotl:property_riserHeight
     ?riserHeightProp ;
   ifcotl:property_treadLength ?treadLenghtProp .
  ?riserHeightProp rdf:value ?riserHeightPropVal .
  ?treadLenghtProp rdf:value ?treadlengthPropVal .
  BIND( ((2*?riserHeightPropVal)+?treadlengthPropVal) AS
     ?value)
  FILTER( (?value >= 570) && (?value <= 630) )
 }
}
```

Lastly, conditional constraints can also be defined using SPARQL, by using an `IF` statement. As shown in Listing 29 the result is set to true if the height difference is between 250 and 500 and the slope is less than 6.25%. The query then filters out all components that comply with the constraint and returns all other ramps.

Listing 29: Conditional constraint using SPARQL

```
SELECT ?invalidObjects ?req
WHERE {
 BIND("Req05_nok" AS ?req) .
 ?invalidObjects a ifcotl:Ramp .
 FILTER NOT EXISTS {
   ?invalidObjects otl:property_heightDifference
     ?heightDifferenceProp ;
     otl:property_slope ?slopeProp .
   ?heightDifferenceProp rdf:value
     ?heightDifferencePropVal .
   ?slopeProp rdf:value ?slopePropVal .
   BIND (
    IF( (?heightDifferencePropVal > 250) &&
     (?heightDifferencePropVal <= 500) && (?slopePropVal <=
     0.0625), true, 1/0)
   AS ?result )
   FILTER (?result = true)
  }
}
```

It is possible to define all five types of proposed constraints with SPARQL, without the need for SPARQL extensions. However, this approach lacks a standardized validation report format, since it was not originally designed for validation use cases. Another aspect is that, while standardized, the results of a query are in practice not guaranteed to be the same due to many aspects that implementers added over time not included in the original specification. This is similar to what happened with relational databases, where the evolving implementations resulted in a myriad of Structured Query Language (SQL) dialects partially limiting portability [57].

4.3.4. Shapes Constraint Language (SHACL)

SHACL [40] is a W3C standard for validating RDF graphs against a set of constraints, which are expressed as shapes. SHACL supports the use of SPARQL queries inside shapes, leading to more expressivity. The result of a SHACL validation process is a standardized RDF graph, which contains the validation report in a machine-readable format. The standardized report makes it easier to interpret the results and retrieve contextual information about the non-complying components in the validated project datasets. Lots of SHACL implementations can be found, however, the support of SHACL-SPARQL is not mandatory. Furthermore, the SHACL standard does not force tool developers to signal when there is a lack of support for this feature, giving the impression that all components are always passing the constraint since it is not actually tested. The user thus needs to verify if the used validator supports this functionality, to ensure a thorough checking procedure. To validate the examples in this section, the pySHACL [58] library was used, which supports the validation of SHACL-SPARQL

constraints. The construction data represented in LBD and the SHACL constraints can be found at OSF:SHACL.[11]

The information availability constraint is defined by targeting the Door class, and defining that the OverallWidth property of this component should have exactly one value, by setting both `minCount` and `maxCount` to one, as shown in Listing 30. The non-complying validation report of this shape is shown in Listing 31. While the result message is quite generic in this example, it can be further specified in the shape using `sh:message`.

Listing 30: Information availability constraint using SHACL

```
reg:DoorShape_req1 a sh:NodeShape ;
  sh:targetClass ifcotl:Door ;
  sh:property reg:Attribute_OverallWidth_req1 .

reg:Attribute_OverallWidth_req1 a sh:PropertyShape ;
  sh:path ifcotl:attribute_OverallWidth ;
  sh:minCount 1 ;
  sh:maxCount 1 .
```

Listing 31: SHACL validation report

```
[
  a sh:ValidationResult ;
  sh:resultSeverity sh:Violation ;
  sh:sourceConstraintComponent
     sh:MinCountConstraintComponent ;
  sh:sourceShape reg:Attribute_OverallWidth_req1 ;
  sh:focusNode
     <https://data.myexample.org/project-HS/id/doorA_fail01>
     ;
  sh:resultPath ifcotl:attribute_OverallWidth ;
  sh:resultMessage "Less than 1 values" ;
] .
```

To define a value constraint, a similar construct as the information availability constraint is used, although the OverallWidth path now has an imposed `minInclusive` value, as shown in Listing 32. Similar constraints can be defined using `maxInclusive`, `minExclusive`, and `maxExclusive`.

Listing 32: Value constraint using SHACL

```
reg:DoorShape_req2 a sh:NodeShape ;
  sh:targetClass ifcotl:Door ;
  sh:property reg:Attribute_OverallWidth_req2 .

reg:Attribute_OverallWidth_req2 a sh:PropertyShape ;
  sh:path (ifcotl:attribute_OverallWidth rdf:value) ;
  sh:minInclusive "900" .
```

SHACL furthermore allows the definition of relational constraints. Listing 33 shows that the stair class is targeted, after which it is defined that this stair should have two subelements of the type railing, using `sh:qualifiedValueShape` and `sh:qualifiedMinCount`. Similarly, `sh:qualifiedMaxCount` can be used for equivalent constraints. It should be noted that this shape does not check the relative placement of the railings to the stair, since the assumption is made that multiple railings cannot be placed on the same side of the stair. Since SHACL, operates under the UNA, like SPARQL, it assumes each railing represents a different railing.

Listing 33: Relational constraint using SHACL

```
reg:StairShape_req3 a sh:NodeShape ;
  sh:targetClass ifcotl:Stair ;
  sh:property [
    sh:path sml:hasPart ;
    sh:qualifiedValueShape [sh:class ifcotl:Railing] ;
    sh:qualifiedMinCount 2
  ] .
```

[11] https://osf.io/j7raf/?view_only=4b225ebc11114eab94bffb261ce29361

Mathematical constraints can be defined by embedding SPARQL queries in the SHACL shape. As Listing 34 shows, the sum of two times the riser plus a tread is bound as the calculated result, after which a filter defines the allowed range of this result. The construct is similar to the one defined using SPARQL (Listing 28), although the filter in this example selects the values outside of the range due to the FILTER NOT EXISTS construct. The prefix binding at the start of the `sh:select` construct is omitted in the Listing 34, for conciseness.

Listing 34: Mathematical constraint using SHACL

```
reg:StairShape_req4 a sh:NodeShape ;
  sh:targetClass ifcotl:Stair ;
  sh:sparql [
    a sh:SPARQLConstraint ;
    sh:select """
      [...]
      SELECT $this ?value
      WHERE {
        $this ifcotl:property_riserHeight ?propHeight .
        ?propHeight rdf:value ?height .

        $this ifcotl:property_treadLength ?propLength .
        ?propLength rdf:value ?length .

        BIND(((2 * ?height) + ?length) AS ?value)
        FILTER( (?value <= 570) || (?value >= 630))
      }
    """
  ] .
```

Conditional constraints can be defined in SHACL using logical operators such as `OR`, `AND`, `NOT`, and `XONE`. As shown in Listing 36, the 'Between' function shown in Listing 35 is needed to evaluate if the height difference lies in a specified range. If this is the case, the ramp should also comply with the second shape defined with the node predicate, due to the `sh:and` construct. This second shape defines that the slope should be less than 6.25%.

Listing 35: SHACL-SPARQL function to evaluate if a value lies in a range

```
reg:Between
 a sh:SPARQLFunction ;
 rdfs:comment "Returns True if op1 < op2 < op3" ;
 sh:parameter [
  sh:path reg:op1 ;
  sh:datatype xsd:double ;
 ] ;
 sh:parameter [
  sh:path reg:op2 ;
  sh:datatype xsd:double ;
 ] ;
 sh:parameter [
  sh:path reg:op3 ;
  sh:datatype xsd:double ;
 ] ;
 sh:returnType xsd:boolean ;
 sh:select """
  SELECT ?result
  WHERE {
   BIND(IF(?op1 < ?op2 && ?op2 < ?op3, true, false) AS
     ?result) .
  }
  """ .
```

Listing 36: Conditional constraint using SHACL

```
reg:RampShape_req5
  a sh:NodeShape ;
  sh:targetClass ifcotl:Ramp ;
  sh:and (
    [sh:expression [
      reg:Between (
        250
        [sh:path (otl:property_heightDifference rdf:value)]
        500
      )
    ]]
```

```
    sh:node reg:SlopeShape_req5
  ) .

reg:SlopeShape_req5
  a sh:NodeShape ;
  sh:property [
    sh:path (otl:property_slope rdf:value) ;
    sh:maxInclusive 0.0625
  ] .
```

SHACL was able to define the five categories of constraints, mainly due to the implementation of SHACL-SPARQL, the extension of SHACL-Core. Furthermore, SHACL has a standardized validation report as output, which is a benefit for use in compliance checking workflows. However, the validation report only contains information about project data that does not conform to the SHACL shapes, meaning it does not explicitly mention which project data has passed certain SHACL constraints. Additionally, low-level functions such as 'Between' are necessary to check the outcome of a SHACL-SPARQL function, which can result in quite complex shapes.

5. Discussion

Several approaches for compliance checking were checked, which can be grouped into three parts:

- Methods operating on IFC data: Solibri Model Checker (a proprietary tool), and the upcoming IDS standard
- General data standards and their accompanying schema definition languages: JSON Schema and XSD
- Linked Data approaches: OWL, SWRL, SPARQL, and SHACL

These approaches were tested against five flavors of rules (information availability, value, relational, mathematical, and conditional constraints), leading to the summary in Fig. 8. In this table, a ● means the approach supports the corresponding requirement category fully, whereas a ○ signifies that the requirement category, to the best of our knowledge, cannot be defined with this approach.

From a functional point of view, all categories of constraints are supported by SMC, XSD, SWRL, SPARQL and SHACL. SMC furthermore includes built-in algorithms to derive alphanumeric data from geometry, while the other approaches require a preprocessing step to extract alphanumeric data from geometrical project data. When the geometrical analysis can be separated from the actual validation step, a certain amount of overhead is initiated while this separation is also expected to result in more flexibility regarding the geometry formats, geometry types (BREP, CSG, meshes, etc.), and geometry modeling conventions.

Secondly, when zooming in on the required input data formats, a separation can be made between methods requiring IFC data (i.e. a dataset conforming to the IFC schema), JSON, XML, and RDF (in any widely supported RDF serialization, e.g. .ttl, .jsonld, .nt, etc.). The approaches requiring IFC data (SMC and IDS) cannot be used to test other kinds of data, as they are specifically made for evaluating 3D construction models in this format. This can be considered a downside compared to JSON Schema, XSD, and Linked Data-based approaches which are applicable in virtually any domain. Consequently, there is a much bigger chance that a software developer will know any of those more general data standards. Furthermore, JSON, XML, and RDF datasets can stem from a plethora of sources (inspection tools, planning, quantity surveying, etc.) while IFC files are only available through a BIM authoring tool.

A third angle for comparison is the degree of openness and standardization for formally expressing requirements, which is important as we aim at a level playing field for compliance checking solutions. Ideally, each organization involved in a project can use the same formal rules in the software application it prefers (e.g. concerning price, support, speed, security, etc.). The only approach that is proprietary in the comparison is SMC. All other methods are based on open specifications, while some of them are even elevated to standards by IETF (JSON Schema) and W3C (XSD, OWL, SPARQL, and SHACL). Note that buildingSMARTs IDS method is still under development at the time of writing and cannot yet be considered a standard.

	information availability	value	relational	mathematical	conditional
SMC	●	●	●	●	●
IDS	●	●	●	○	○
XSD	●	●	●	●	●
JSON schema	●	●	●	○	●
OWL (+SPARQL)	●	●	●	○	●
SWRL (+SPARQL)	●	●	●	●	●
SPARQL	●	●	●	●	●
SHACL	●	●	●	●	●

Fig. 8. Summary of the approaches.

Finally, if we also investigate the subject of standardized outputs of a validation process, we noticed that, to the best of our knowledge, only the JSON Schema, SPARQL, and SHACL specification define a standard output format, where the JSON Schema and SHACL outputs are in origin more oriented to the validation use cases. In the case of SPARQL CONSTRUCT queries or SHACL, the report can thus be queried together with the original project data and formal requirements, to add additional context to humans. While this paper focused on automating the compliance checking procedure, it is (for now) unimaginable to create a fully automated checking workflow, since not all parts of the regulation can be translated into computer-interpretable rules. Even in the automated checking workflow, not only the accordance of the written building legislation and the machine-readable rules will have to be reviewed by a human in the loop, but also the dataset quality. For quality improvements (e.g. quality of classification), approaches based on Machine Learning could help a human reviewer to faster get an idea of the quality of the data. For the harmonization, there is a need for defining shared ontologies and/or alignments as well as preprocessing procedures for e.g. units. In contrast to the rigid structure of IFC and the unlimited structures of XML and JSON Schema, Linked Data-based approaches can rely on the recent EN17632-1:2022 which defines basic graph patterns for relations and dataset modeling.

6. Conclusion

This paper compares different approaches to capture requirements derived from building regulations as machine-readable constraints, for use in ACC. The different approaches are evaluated by defining constraints for the same set of five requirements with each approach. The approaches can be distinguished as (1) methods operating on IFC data, (2) general data standards, and (3) Linked Data approaches. While IFC is commonly used in the AECO industry, it is not widely implemented outside of this industry. This leads to less rapid development of the IFC-based approaches. However, the opposite is true for general data standards and Linked Data approaches. Since they are used in various

domains, the approaches are generally better documented and more implementations are available. Out of the eight approaches that were investigated, the Linked Data-based SHACL approach was concluded to be the best suited for use in compliance checking in the AECO industry. The most apparent perk is that it supports a standardized validation report, making it possible to implement it in a fully automated construction design workflow. However, the proposed methodology has some limitations: first, we did not investigate the ease of creating constraints by regulation authors or designers. Furthermore, direct validation of geometries was out of scope, assuming alphanumeric properties were derived using a separate process. Although there are limitations to this research, we hope the provided listings make it easier for future research to test, compare, and implement these approaches.

CRediT authorship contribution statement

Emma Nuyts: Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Mathias Bonduel:** Conceptualization, Data curation, Funding acquisition, Investigation, Methodology, Project administration, Software, Validation, Writing – review & editing. **Ruben Verstraeten:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All data related to this article is publicly available at OSF[12] under the *CC-By Attribution 4.0* license.

Acknowledgments

This work was supported by the European Union's Horizon Research and Innovation Program - Digichecks [grant number 101058541].

References

[1] R. Amor, J. Dimyadi, The promise of automated compliance checking, Devel. Built Environ. 5 (2021) 100039, http://dx.doi.org/10.1016/j.dibe.2020.100039.
[2] A. Mowbray, P. Chung, G. Greenleaf, Representing legislative Rules as Code: Reducing the problems of 'scaling up', Comput. Law Secur. Rev. 48 (2023) 105772, http://dx.doi.org/10.1016/j.clsr.2022.105772.
[3] J. Zhang, N.M. El-Gohary, Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking, Autom. Constr. 73 (2017) 45–57, http://dx.doi.org/10.1016/j.autcon.2016.08.027.
[4] Y.-C. Zhou, Z. Zheng, J.-R. Lin, X.-Z. Lu, Integrating NLP and context-free grammar for complex rule interpretation towards automated compliance checking, Comput. Ind. 142 (2022) 103746, http://dx.doi.org/10.1016/j.compind.2022.103746.
[5] ISO 16739-1:2018 - Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries, 2018.
[6] S. Daum, A. Borrmann, Processing of topological BIM queries using boundary representation based methods, Adv. Eng. Inform. 28 (4) (2014) 272–286, http://dx.doi.org/10.1016/j.aei.2014.06.001.
[7] W. Solihin, J. Dimyadi, Y.-C. Lee, C. Eastman, R. Amor, The critical role of accessible data for BIM-based automated rule checking systems, in: Lean and Computing in Construction Congress - Volume 1: Proceedings of the Joint Conference on Computing in Construction, Heriot-Watt University, Heraklion, Crete, Greece, 2017, pp. 53–60, http://dx.doi.org/10.24928/JC3-2017/0161.
[8] L. Ding, R. Drogemuller, M. Rosenman, D. Marchant, J. Gero, Automating code checking for building designs: DesignCheck, in: K. Brown, K. Hampson, P.S. Brandon (Eds.), Clients Driving Construction Innovation: Moving Ideas Into Practice, Cooperative Research Centre for Construction Innovation, Brisbane, Qld., Australia, 2006.
[9] B.-H. Goh, E-government for construction: The case of Singapore's CORENET project, in: L.D. Xu, A.M. Tjoa, S.S. Chaudhry (Eds.), Research and Practical Issues of Enterprise Information Systems II Volume 1, Vol. 254, Springer US, Boston, MA, 2008, pp. 327–336, http://dx.doi.org/10.1007/978-0-387-75902-9_34, Series Title: IFIP — The International Federation for Information Processing.
[10] E. Hjelseth, Public BIM-based model checking solutions: lessons learned from Singapore and Norway, Bristol, UK, 2015, pp. 421–436, http://dx.doi.org/10.2495/BIM150351.
[11] Solibri, 2023, URL https://www.solibri.com/. Last visited 2023-08-05.
[12] J. Wix, N. Nisbet, T. Liebich, Using constraints to validate and check building information models, in: A. Zarli, R. Scherer (Eds.), EWork and EBusiness in Architecture, Engineering and Construction, CRC Press, 2008, http://dx.doi.org/10.1201/9780203883327.
[13] buildingSMART, Model View Definitions (MVD), buildingSMART Technical, URL https://technical.buildingsmart.org/standards/ifc/mvd/. Last visited 2023-07-04.
[14] T. Chipman, T. Liebich, M. Weise, mvdXML, 2016.
[15] Y.-C. Lee, C.M. Eastman, W. Solihin, Logic for ensuring the data exchange integrity of building information models, Autom. Constr. 85 (2018) 249–262, http://dx.doi.org/10.1016/j.autcon.2017.08.010.
[16] buildingSMART, IfcDoc: IFC doc toolkit, 2022, GitHub. URL https://github.com/buildingsmart-private/IfcDoc. Last visited 2023-08-27.
[17] L. Van Berlo, Belangrijke update: MVD's worden Vervangen door specificaties voor informatielevering (IDS), 2023, buildingSMART Nederland. URL https://www.buildingsmart.nl/nieuws/2023-02-07-belangrijke-update-mvds-worden-vervangen-door-specificaties-voor-informatielevering-ids. Last visited 2023-08-06.
[18] A. Tomczak, L. Berlo, T. Krijnen, A. Borrmann, M. Bolpagni, A review of methods to specify information requirements in digital construction projects, in: IOP Conference Series: Earth and Environmental Science, Volume 1101, W078: Information Technology for Construction, IOP Publishing Ltd, 2022, http://dx.doi.org/10.1088/1755-1315/1101/9/092024, 092024.
[19] C. Preidel, S. Daum, A. Borrmann, Data retrieval from building information models based on visual programming, Vis. Eng. 5 (1) (2017) 18, http://dx.doi.org/10.1186/s40327-017-0055-0.
[20] J. Dimyadi, W. Solihin, C. Eastman, R. Amor, Integrating the BIM rule language into compliant design audit processes, in: Proceedings of 33rd CIB W78 Conference: IT in Construction, Brisbane, Australia, 2016, pp. 1–10, Issue: November.
[21] Y. Shafranovich, Common Format and MIME Type for Comma-Separated Values (CSV) Files, Tech. Rep., 2005, http://dx.doi.org/10.17487/rfc4180, RFC4180. RFC Editor.
[22] W3C, Metadata vocabulary for tabular data, 2015, URL https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/. 2023-08-29.
[23] CSV schema, 2022, URL https://digital-preservation.github.io/csv-schema/. Last visited 2023-08-29.
[24] F. Standards, Table schema, 2023, URL https://specs.frictionlessdata.io//table-schema/. Last visited 2023-08-29.
[25] W3C, XML schema definition language (XSD) 1.1 Part 1: Structures, 2012, URL https://www.w3.org/TR/xmlschema11-1/. Last visited 2023-07-04.
[26] W3C, XML schema definition language (XSD) 1.1 Part 2: Datatypes, 2012, URL https://www.w3.org/TR/xmlschema11-2/. Last visited 2023-07-04.
[27] W3C, XML schema Part 1: Structures second edition, 2004, URL https://www.w3.org/TR/xmlschema-1/. Last visited 2023-08-05.
[28] Ecma, ECMA-404: The JSON data interchange syntax, 2017, Ecma International. URL https://www.ecma-international.org/publications-and-standards/standards/ecma-404/. Last visited 2023-07-04.
[29] JSON schema, 2020, JSON Schema. URL https://json-schema.org/. Last visited 2023-08-06.
[30] W3C, RDF 1.1 concepts and abstract syntax, 2014, URL https://www.w3.org/TR/rdf11-concepts/. 2023-08-29.
[31] W3C, OWL 2 web ontology language document overview (second edition), 2012, URL https://www.w3.org/TR/owl2-overview/. Last visited 2023-07-25.
[32] J. Tao, E. Sirin, J. Bao, D. McGuinness, Integrity constraints in OWL, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 24, (1) 2010, pp. 1443–1448, http://dx.doi.org/10.1609/aaai.v24i1.7525.
[33] W3C, SWRL: A semantic web rule language combining OWL and RuleML, 2004, URL https://www.w3.org/Submission/SWRL/. Last Visited 2023-07-04.
[34] M. Uhm, G. Lee, Y. Park, S. Kim, J. Jung, J.-k. Lee, Requirements for computational rule checking of requests for proposals (RFPs) for building designs in South Korea, Adv. Eng. Inform. 29 (3) (2015) 602–615, http://dx.doi.org/10.1016/j.aei.2015.05.006.
[35] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, J. Hendler, N3Logic: A logical framework for the world wide web, Theory Pract. Logic Program. (2007) http://dx.doi.org/10.48550/ARXIV.0711.1533, Publisher: arXiv Version Number: 1.
[36] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van De Walle, J. Van Campenhout, A semantic rule checking environment for building performance checking, Autom. Constr. 20 (5) (2011) 506–518, http://dx.doi.org/10.1016/j.autcon.2010.11.017.
[37] W3C, SPARQL 1.1 query language, 2013, URL https://www.w3.org/TR/sparql11-query/. 2023-07-04.

[12] https://osf.io/5rwt6/?view_only=7637d2f2718341a59a5bd6635965e973

[38] B. Zhong, C. Gan, H. Luo, X. Xing, Ontology-based framework for building environmental monitoring and compliance checking under BIM environment, Build. Environ. 141 (2018) 127–142, http://dx.doi.org/10.1016/j.buildenv.2018.05.046.

[39] Z. Zheng, Y.-C. Zhou, X.-Z. Lu, J.-R. Lin, Knowledge-informed semantic alignment and rule interpretation for automated compliance checking, Autom. Constr. 142 (2022) 104524, http://dx.doi.org/10.1016/j.autcon.2022.104524.

[40] W3C, Shapes constraint language (SHACL), 2017, URL https://www.w3.org/TR/shacl/. 2023-07-04.

[41] R.K. Soman, M. Molina-Solana, J.K. Whyte, Linked-data based constraint-checking (LDCC) to support look-ahead planning in construction, Autom. Constr. 120 (2020) 103369, http://dx.doi.org/10.1016/j.autcon.2020.103369, Publisher: Elsevier.

[42] W3C, ShEx - shape expressions, 2019, URL https://shex.io/. 2023-08-05.

[43] P. Pauwels, S. Zhang, Semantic rule-checking for regulation compliance checking: An overview of strategies and approaches, in: Proceedings of the 32nd CIB W78 Conference, Eindhoven, Netherlands, 2015, pp. 619–628.

[44] Besluit van de Vlaamse Regering tot vaststelling van een gewestelijke stedenbouwkundige verordening betreffende toegankelijkheid, 2009, URL https://www.toegankelijkgebouw.be/Regelgeving/Downloads/tabid/328/Default.aspx.

[45] M. Bonduel, A. Wagner, P. Pauwels, M. Vergauwen, R. Klein, Including widespread geometry schemas into Linked Data-based BIM applied to built heritage, Proc. Inst. Civ. Eng. - Smart Infrastruct. Construct. 172 (1) (2019) 34–51, http://dx.doi.org/10.1680/jsmic.19.00014.

[46] NBN EN 17632-1:2022 Building information modelling (BIM). Semantic modelling and linking (SML). Generic modelling patterns, 2022.

[47] P. Pauwels, IFCtoRDF, 2023, GitHub. URL https://github.com/pipauwel/IFCtoRDF. Last visited 2023-08-27.

[48] J. Oraskari, IFCtoLBD, 2023, URL https://github.com/jyrkioraskari/IFCtoLBD. Last visited 2023-08-27.

[49] buildingSMART, Information Delivery Specification IDS, buildingSMART Technical, 2020, URL https://technical.buildingsmart.org/projects/information-delivery-specification-ids/. Last visited 2023-07-04.

[50] Accasoftware, IDS BIM, 2023, URL https://www.accasoftware.com/en/information-delivery-specification-ids. Last visited 2023-08-16.

[51] buildingSMART, Advanced information requirements, 2022, URL https://github.com/buildingSMART/IDS/blob/master/Documentation/specifications.md#advanced-information-requirements.

[52] XML Schema validation service, 2022, URL https://www.softwarebytes.org/xmlvalidation/. Last visited 2023-08-16.

[53] Apache, Xerces2 Java XML parser readme, 2023, URL https://xerces.apache.org/xerces2-j/. Last visited 2023-08-27.

[54] Ajv JSON schema validator, 2021, URL https://ajv.js.org/. Last visited 2023-08-16.

[55] Swrltab-plugin, 2023, Protégé Project. URL https://github.com/protegeproject/swrltab-plugin. original-date: 2014-06-29.

[56] Protege desktop, 2023, URL https://github.com/protegeproject/protege. Last visited 2023-08-27.

[57] D. Chamberlin, SQL, in: L. Liu, M.T. Özsu (Eds.), Encyclopedia of Database Systems, Springer US, Boston, MA, 2009, pp. 2753–2760, http://dx.doi.org/10.1007/978-0-387-39940-9_1091.

[58] A. Sommer, N. Car, pySHACL, 2023, http://dx.doi.org/10.5281/ZENODO.4750840, Zenodo.