Validation of building models against legislation using SHACL

Emma Nuyts[1,*], Jeroen Werbrouck[1], Ruben Verstraeten[1] and Louise Deprez[1]

[1]*Department of Architecture and Urban Planning, Ghent University, Ghent, Belgium*

Abstract

Building information is commonly available in a machine-readable format, whereas normative knowledge is represented in a human-readable way, making human intervention mandatory. A solution for this manual checking procedure is Automated Compliance Checking (ACC). Commercial systems rely on hard-coded requirements, which are almost as error-prone and time-consuming as manually checking the compliance, or focus solely on one specific software package. Hence, this research focuses on Semantic Web technologies to enable a faster and more transparent rule-checking process. The requirements from the legislation will be defined using the Shapes Constraint Language (SHACL). This language facilitates defining constraints in the Terse RDF Triple Language (Turtle), making one set of constraints both human- and machine-readable. Additionally, SHACL enables compliance checking to be combined with quality constraints, ensuring that all necessary data is present in the building project. The proposed methodology is applicable to any type of prescriptive building legislation, provided that the data is defined in the building graph.

Keywords

Automated Compliance Checking, Linked Data, SHACL

1. Introduction

The use of Building Information Management (BIM) has seen a significant increase within the Architecture Engineering and Construction (AEC) industry in recent years. However, the full potential of BIM models is often not achieved. Specifically, compliance with building codes remains a manual process, by retrieving information from the BIM model and reviewing this data separately. This approach is not only time-consuming but also error-prone for both architects and building permit reviewers. This issue is confirmed by research of 2019 by the Flemish institute for accessibility Inter, which found that out of 147 reviewed building permits, only 9 were designed complying with the regulation on accessibility [1]. One way to overcome these challenges is by implementing Automated Compliance Checking (ACC). This process utilizes the BIM data directly to evaluate compliance, ensuring a more accurate checking procedure. Furthermore, the use of ACC allows more frequent evaluations throughout the design process, providing feedback early on in the process and ultimately leading to more cost-efficient designs.

Proceedings of the 11th Linked Data in Architecture and Construction Workshop - LDAC2023, June 15–16, 2023, Matera, Italy

*Corresponding author

emma.nuyts@ugent.be (E. Nuyts); jeroen.werbrouck@ugent.be (J. Werbrouck); ruben.verstraeten@ugent.be (R. Verstraeten); louise.deprez@ugent.be (L. Deprez)

2. Related work

2.1. Semantic Web and Linked Data

In 1990, Tim Berners-Lee envisioned the use of the World Wide Web as one central data storage platform for use in CERN [2]. However, the Web evolved from a static and informative Web 1.0 to an interactive Web 2.0. While this version of the Web mainly focused on human user experience, a new development has taken place with Web 3.0: this most recent version entails an integrated web experience where machines can understand data in a similar way as humans [3]. This version of the Web is often referred to as the Semantic Web, of which the concept of Linked Data lies at its core. The World Wide Web Consortium (W3C) defines Linked Data as '*the collection of interrelated datasets on the Web*', which is used for large-scale integration of, and reasoning on, data across the Web [4]. To achieve full interoperability, the machine-readable description is standardized in the Resource Description Framework (RDF). Furthermore, Semantic Web technologies like the SPARQL Protocol and RDF Query Language (SPARQL) [5] and the Shapes Constraints Language (SHACL) [6] can be used to respectively query and validate RDF graphs.

2.2. Linked Data in the AEC industry

While the open and neutral file-format Industry Foundation Classes (IFC) has improved interoperability in the AEC industry, it has some downsides when used in the context of the Semantic Web: (a) the used EXPRESS and XSD languages lack methods for defining formal semantics, making it difficult for reasoning and querying purposes, (b) it is complex to link the building data stored in an IFC file to related data on the web and (c) the IFC schema is large and rather complex, making it a challenge to implement it correctly [7].

To address these limitations, new methods were developed with the Semantic Web in mind, like ifcOWL and Linked Building Data (LBD). ifcOWL was developed as an ontology equivalent to the original IFC schema but made available as a Semantic Web ontology. However, since this ontology lies close to the original schema and contains a large amount of information, it has some negative consequences: many of the EXPRESS-specific semantic constructs are maintained and result in complex and counterintuitive constructs, and the resulting graphs are at least as complex and large as the original IFC model [8]. Since the AEC industry needed a simplified method for handling IFC models, LBD was introduced. For this, the IFC file is first converted to ifcOWL, after which the topological ifcOWL classes are mapped to the related Building Topology Ontology (BOT) classes [9]. Furthermore, the LBD Community Group proposes three levels of modeling complexity (L1-L3) for properties. The complexity is defined depending on the number of steps needed to navigate from an element to its property via the RDF graph [7].

2.3. Steps in the ACC process

2.3.1. Interpretation of normative knowledge

To use the normative data for compliance checking, it must first be processed. There are a few methods for doing this, such as using Natural Language Processing (NLP), mark-up language, or proposing a new methodology or framework.

There is ongoing research [10] on how deep learning NLP techniques can extract Decision Model and Notation (DMN) from text. The Object Management Group (OMG) introduced DMN as a tool to model, communicate and execute operational decisions [11] and to be able to define different criteria and present decision options more clearly than the Business Process Model and Notation (BPMN) standard. These visual programming Languages have the advantage that the checking process can be displayed graphically, which significantly increases readability [12]. Furthermore, research [12, 13] has shown BPMN and DMN are promising approaches for the representation of guideline content. Especially since these notations are already used in the AEC industry in the context of Information Delivery Manuals (IDMs).

Another option for processing normative data is using mark-up language, such as Requirement Applicabilities Selection and Exceptions (RASE) [14]. This is a promising methodology for use in the AEC industry. This concept was specifically developed to transform normative documents into a set of well-defined constraints which can be implemented in model-checking software. RASE uses four tags with corresponding colors to differentiate sentences in legislation. However, the biggest limitation is that the mark-up is done manually, meaning it is still laborious and error-prone. The efficiency of the mark-up also heavily relies on sentence structures.

A final possibility for processing normative data is by proposing a new methodology or framework. This approach involves updating the building legislation to a machine-readable format, by using a restricted set of grammar rules and vocabulary. These restrictions reduce the ambiguity and complexity of natural language and improve human readability and machine processability [15].
Since the main scope of this paper is to evaluate SHACL for use in compliance checking, and not the full automation of the process, the manual RASE mark-up language will be used in this paper to structure the normative data.

2.3.2. Mapping applicabilities to ontologies

Each article of the regulation applies to a building element, called the 'applicability' of that article. These applicabilities need to be mapped to the corresponding definition in the used ontologies, to be able to create machine-readable constraints. For example, a *stair* with properties like a *riser height* and *tread length* need to be mapped to a *beo:StairFlight* with a *props:riserHeightIfcStairFlight* and a *props:treadLengthIfcStairFlight*. This can be done automatically, for example using the npm package nlp.js [16], as demonstrated in the LD-BIM web application developed by Rasmussen & Schlachter [17], where a user can search for *doors*, and the webpage highlights all *ifc:IfcDoors* in the 3D building model. However, such automatic mapping of terms lies out of scope of this paper, hence it will be done manually.

2.3.3. Creating machine-readable constraints

After processing the normative knowledge, these requirements need to be converted into machine-readable constraints. Pauwels & Zhang [18] have defined three strategies for this conversion: using hard-coded requirements, rule-checking by querying or using a dedicated rule language such as the Semantic Web Rule Language (SWRL), N3Logic or the Drools Rule Language (DRL). However, another method could be applied, by using the Shapes Constraint

Language (SHACL). While this language was originally developed to check the completeness of graphs [19], it can also be used to check the contents of the graph. Using SHACL for compliance checking has been researched previously: Oraskari et al. [20] showed that SHACL allows checking procedures on building models converted to LBD ontologies, by creating MVD views ensuring the expected information is present in the graph. Furthermore, Elshani et al. [21] argue that augmenting Building and Habitats object Model (BHoM) Knowledge Bases (KBs) with Semantic Web rules, like SWRL or SHACL, would increase the usage of KBs in the AEC industry. Moreover, Zentgraf et al. [22] demonstrate how quality assurance of properties can be conducted using SHACL shapes. Lastly, Kovacs & Micsik [23] give an example of using SHACL for compliance checking, by showing the shape for checking the thermal transmittance of a window.

3. Building validation using SHACL

3.1. Research Question

While the use of SHACL for compliance checking has been evaluated in existing literature, the examples provided have focused on relatively simple constraints. This study aims to assess the suitability of SHACL for more complex constraints, including those that involve checking relationships between building elements, performing calculations, and creating conditional statements. The main research question is therefore: *Can SHACL be used to evaluate more complex constraints for compliance checking purposes?*
In this paper, three different types of constraints (value, relational and mathematical) will be proposed in the Terse RDF Triple Language (Turtle) format, as well as methods for combining these different types. The illustrative examples in this paper will make use of the prefixes listed in Listing 1 and are made available on GitHub[1]. The examples will assume an L2 modeling complexity of the Linked Building Data graph, as proposed in Bonduel et al. [7]. It is important to note that this methodology focuses on the semantic information of a building, rather than its geometric representation. For example, a ramp will be characterized by its numerical height and length, leading to a calculated slope, rather than determining the geometrical angle between the sloped surface and the ground plane.

Listing 1: Prefixes used in this paper

```
@prefix beo: <https://pi.pauwel.be/voc/buildingelement#> .
@prefix bot: <https://w3id.org/bot#> .
@prefix ex: <https://example.org#> .
@prefix props: <https://w3id.org/props#> .
@prefix schema: <https://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

[1]https://github.com/NuytsE/Validation-of-building-models-against-legislation-using-SHACL

3.2. Use case on accessibility

The Flemish regulation on accessibility is chosen as a use case for this paper, not only because of the striking results of Inter [1] but also because it uses all types of constraints proposed in this paper. However, the constraints can be applied to each kind of prescriptive building code. The three types of constraints will be created using the steps explained in Section 2: the normative knowledge will first be structured using the RASE mark-up language, after which the applicabilities will be mapped to the corresponding classes and properties defined in the ontology. The last step is to create machine-readable constraints using SHACL.

3.3. Ensuring model quality

To ensure all the necessary data is available in the building graph, thus prohibiting 'false compliance', the Information Delivery Specification (IDS) can be used. This computer-interpretable document defines Exchange Requirements of model-based exchange, defining how objects, classifications, properties, and units need to be delivered and exchanged [24]. However, since this standard is not fully implemented in the AEC industry, it is defined that every path used in the SHACL shapes should have exactly one value. In the case of the regulation on accessibility, we want to ensure that each door has a specified door height. This property should occur exactly one time since a door with either zero or more than one door height is not desired, as shown in Listing 2.

Listing 2: Example of a quality constraint

```
ex:DoorProperties
  a sh:NodeShape ; #apply the shape to a focus node
  sh:targetClass beo:Door ; #target all nodes with class 'Door'
  sh:property [ #target a property of each 'Door'
    sh:path props:overallHeightIfcDoor ; #target the height predicate
    sh:property ex:DoorHeightProperty ; #name the object of this predicate path
    sh:minCount 1 ; #each 'Door' should have at least one height property
    sh:maxCount 1 ; #each 'Door' should have at most one height property
    sh:message "Each door should have exactly one overallHeightIfcDoor" ; ] .

ex:DoorHeightProperty
  a sh:PropertyShape ; #target a property of the focus node
  sh:path schema:value ; #target the value predicate
  sh:minCount 1 ; #each 'overallHeightIfcDoor' property should have at least one value
  sh:maxCount 1 ; #each 'overallHeightIfcDoor' property should have at most one value
  sh:message "Each doorheight should have exactly one value" .
```

3.4. Value constraints

The first type of constraint is a so-called value constraint. This type allows checking if a property is more/less than or equal to a given value. First, the interpretation of the normative knowledge is done using the RASE mark-up on a (translated) paragraph of the Flemish regulation on accessibility, shown in Listing 3. The second step is to map the applicabilities of the regulation to the ontologies used in the building graph. Table 1 shows the corresponding classes and properties of the running example.

Listing 3: Article 22 §1 of the Flemish regulation on accessibility

```
For <a>entrances or doorways</a>, a <a>clear passage height</a> of <r>at least 2.09
    meters</r> must be guaranteed after finishing.
```

Table 1
Mapping applicabilities of the regulation on accessibility to the corresponding classes and properties of the BEO ontology

Article	Type	Applicability	Ontology
Art.22 §1	class	entrances or doorways	beo:Door
	property	clear passage height	props:overallHeightIfcDoor
Art.20 §4	class	stair	beo:Stair
	class	railing	beo:Railing
Art.20 §3	property	riser	props:riserHeightIfcStairFlight
	property	tread	props:treadLengthIfcStairFlight

The last step is to create a machine-readable rule, using SHACL. As shown in Listing 4, the shape targets the class and property derived from the normative data and specifies that this door height should be at least 2.09 meters, using *sh:minInclusive*. For other examples of value constraints, *sh:minExclusive*, *sh:maxInclusive*, or *sh:maxExclusive* can be used in an analogous way. This type of constraint has the limitation that the units of measurement in the SHACL shape and the building data should be the same, to prohibit false compliance. To overcome this challenge, the units can be checked in a quality constraint, or the general agreement is made that the International System of Units (SI) is used.

Listing 4: Example of a value constraint

```
ex:Door
  a sh:NodeShape ; #apply the shape to a focus node
  sh:targetClass beo:Door ; #target all nodes with class 'Door'
  sh:property [ #target a property of each 'Door'
    sh:path props:overallHeightIfcDoor ; #target the height predicate
    sh:property ex:DoorHeight ; ] . #name the object of this predicate path

ex:DoorHeight #the named object is now a subject
  a sh:PropertyShape ; #target a property of the focus node
  sh:path schema:value ; #target the value predicate
  sh:minInclusive "2.09"^^xsd:double ; #the object should be more than 2.09 m
  sh:message "Art.22.1: For entrances or doorways, a clear passage height of at least
      2.09 meters must be guaranteed after finishing" .
```

3.5. Relational constraints

The second type of constraint is a relational constraint. Relational constraints are defined to check if a building element is present in relation to another building element in the project. The interpretation of the normative knowledge using the RASE mark-up language is shown in Listing 5. The second step is to map the applicabilities to classes and properties defined in the ontology, as shown in Table 1.

Listing 5: Article 20 §4 of the Flemish regulation on accessibility

```
A <a>railing</a> should be fitted on <r>both sides</r> of the <a>stair</a> ...
```

The third step is to create a machine-readable SHACL shape. Listing 6 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* en *sh:qualifiedMinCount*. Similarly, *sh:qualifiedMaxCount* can be used for other paragraphs of the regulation. It should be noted that this shape does not check the relative placement of the railings and the stair. The assumption is made that multiple railings cannot be placed on the same side of the stair.

Listing 6: Example of a relational constraint

```
ex:Stair
  a sh:NodeShape ; #apply the shape to a focus node
  sh:targetClass beo:Stair ; #target all nodes with class 'Stair'
  sh:property [ #target a property of each 'Stair'
    sh:path bot:hasSubElement ; #target the subelement predicate
    sh:qualifiedValueShape [sh:class beo:Railing] ; #target the class 'Railing'
    sh:qualifiedMinCount 2 ; #two elements of this class should be present
    sh:message "Art.20.4: A railing should be fitted on both sides of the stair" ; ] .
```

3.6. Mathematical constraints

The last type of constraint is a mathematical constraint, which uses SHACL-SPARQL, the extension of SHACL-Core. This type allows the evaluation of formulas, of which the result can be checked in a separate value constraint. The interpretation of the normative knowledge is shown in Listing 7. The second step is to map the applicabilities to classes and properties defined in the ontology, as shown in Table 1.

Listing 7: Article 20 §3 of the Flemish regulation on accessibility

```
... the <r>sum of twice the <a>riser</a> and once the <a>tread</a> of each step must
    be between 57 cm and 63 cm</r> ...
```

The last step is creating the SHACL shape. A mathematical constraint is created using a *sh:SPARQLFunction*, which consists of a declaration of the parameters, which are the riser and the tread in this case, and a return type. Listing 8 shows the shape of the running example.

Listing 8: Example of a mathematical constraint

```
ex:StairFormula
  a sh:SPARQLFunction ; #define a function
  sh:parameter [ #declare the first parameter
    sh:path ex:riser ; #first parameter is the riser height
    sh:datatype xsd:double ; #the riser is a decimal
  ] ;
  sh:parameter [ #declare the second parameter
    sh:path ex:tread ; #second parameter is the tread length
    sh:datatype xsd:double ; #the tread is a decimal
  ] ;
  sh:returnType xsd:double ; #the returned value is a decimal
```

```
 sh:select """ #start a SPARQL select query
  SELECT ( (2 * $riser + $tread) AS ?result)
  WHERE {
  }
  """ .
```

The function can now be used in a value constraint, to evaluate the result, as shown in Listing 9. This is done by a *sh:expression* and specifying the function parameters in the same order as they were defined in the creation of the SPARQLfunction. For this example, an additional function 'LessThan' needs to be implemented.

Listing 9: Example of a value constraint using predefined functions

```
ex:StairSlope
 a sh:NodeShape ; #apply the shape to a focus node
 sh:targetClass beo:StairFlight ; #target all nodes with class 'StairFlight'
 sh:message "Art.20.3: the sum of twice the riser and once the tread of each step
     must be higher than 0.57 m." ;
 sh:expression [
  ex:LessThan ( #refer to the LessThan(a,b) function
     0.57 #first parameter of LessThan
     [ #second parameter of LessThan
        ex:StairFormula ( #refer to the StairFormula(riser, tread) function
           [sh:path (props:riserHeightIfcStairFlight schema:value)]
           [sh:path (props:treadLengthIfcStairFlight schema:value)])])] .
```

3.7. Conditional statements

To be able to combine the constraints, some conditional statements (if...then...) have to be created. In SHACL, this can be mainly done using *sh:node*, which enables defining that each value node conforms to a given node shape. To properly define the conditional statements, some logical constraint components like *sh:and*, *sh:not*, *sh:or* and *sh:xone* are also needed. Listing 10 shows a conditional statement of the regulation on accessibility, defining that if the slope is less than 10%, an additional constraint should also be fulfilled.

Listing 10: Example of a conditional statement

```
ex:Slopes
 a sh:NodeShape ; #apply the shape to a focus node
 sh:targetClass beo:RampFlight ; #target all nodes with class 'Rampflight'
 sh:message "Art.19.1: The slope of a rampflight is maximally ten percent for level
     differences of up to 0.10 m" ;
 sh:and ( #both constraints should be fulfilled
  [sh:expression [ #first constraint: the slope is less than 10%
     ex:LessThan (
        [ex:Slope (
           [sh:path (props:heightIfcRampFlight schema:value)]
           [sh:path (props:lengthIfcRampFlight schema:value)])]
        10)]]
  sh:node ex:SlopeConstraint) . #if the slope is less than 10%, check the next
      constraint
```

4. Proof of concept

4.1. Conversion workflow

To be able to check the compliance of the building design, the IFC file should be converted to an RDF graph. To accomplish this, different converters exist, like IFCtoLBD [25]. This converter first transforms the file to a temporary ifcOWL graph, which is then converted to an LBD graph. The converter follows the graph traversal, using path templates. The newly created instances get the right LBD OWL classes by using LBD modular ontologies like BOT, PRODUCT, and PROPS. If a higher modeling complexity is chosen by the user, new instance nodes for the properties are created [7].

4.2. Implementation

A proof of concept application is created using npm and pySHACL [26], since there is no JavaScript implementation of SHACL-SPARQL available at the time of writing. The User Interface (UI) is created using React, where a user can upload an LBD graph in Turtle syntax, after which it is evaluated against the rulebook, containing both quality and accessibility shapes. Figure 1 shows the validation report, where transparency to the user is key: apart from the message containing the building legislation, both the source shape and the focus node are returned, meaning it is easily checked exactly which building element does not comply with which shape. Moreover, the identifier of the element is returned, ensuring that the building element can be easily found and altered in the native modeling software.

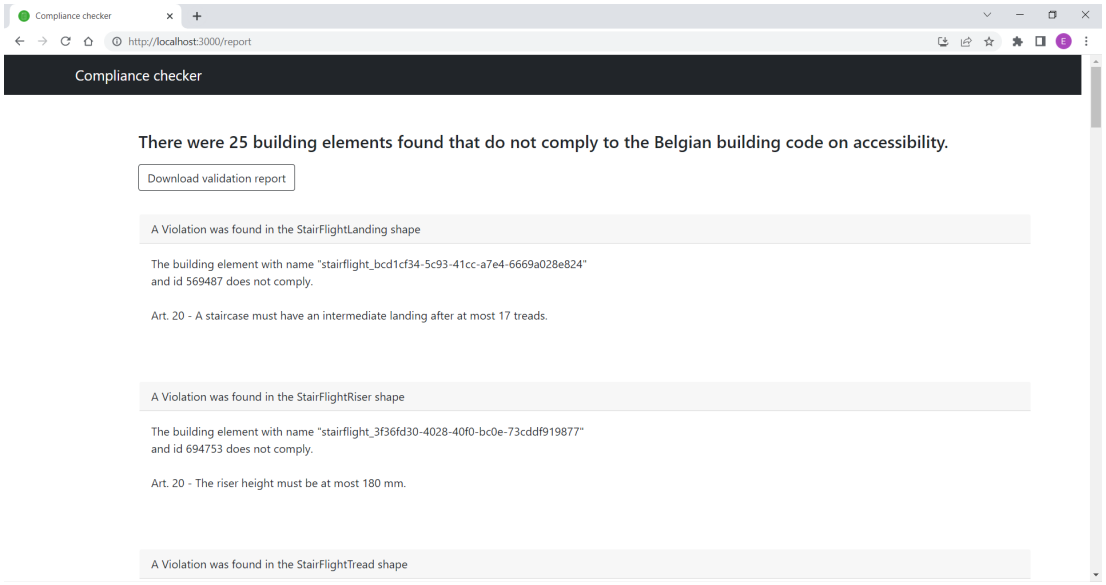

Figure 1: User interface of the validation report

5. Discussion

5.1. Evaluation of SHACL for use in compliance checking

To evaluate the suitability of SHACL for use in compliance checking, the four key requirements for automated code checking as defined by Greenwood et al. [27] can be used:

- The computer-programmed rules must be easily understood by the regulation authors;
- The lifecycle of the rule base must be independent of software and schema updates;
- All development must comply with Open Standards;
- Consideration must be given to the industry processes of model authoring.

While the first requirement is quite subjective, the Turtle syntax is generally seen as the most human-readable syntax for RDF [28]. However, regulation authors without any knowledge of programming, SHACL, or the ontology definitions will not understand this. Though, while the regulation is specifically written to be understandable by architects (and is apparently not), this rulebook is written to check compliance automatically. The user will have some understanding of what is being checked, but a full understanding of the underlying structures is less necessary. The RDF shapes can however be converted into a human-friendly interface if needed. The second requirement is achieved by implementing LBD, which is independent of software packages since it is derived from the open and neutral IFC schema. The third requirement is fulfilled by using Semantic Web technology like SHACL, which is a W3C recommendation. The last requirement is partially avoided by implementing quality constraints. This does not improve the quality of the building model but does check if all the necessary data is present before applying the constraints.

5.2. Limitations and future work

When using SHACL for compliance checking, the main challenge is that the shapes are dependent on the level of modeling complexity of the RDF graph. This means that the same constraint can result in three different SHACL shapes, depending on the level of complexity (L1-L3). Additionally, low-level functions like 'LessThan' are necessary to check the outcome of a SHACL-SPARQL function, which can result in complex value constraints. It is uncertain if this method will be scalable for more complex regulations. Furthermore, the proposed methodology is only suitable for evaluating prescriptive regulations, meaning that it may not be appropriate for assessing acoustical codes, daylight regulations, or energy calculations. It is important to note that a consistent unit system must be used in both the regulation and the building graph to ensure accurate compliance checking.

Possible future work can be done on developing a methodology for checking the compliance of geometry or the relative positioning of building elements, whereas this paper focused on the properties defined in the semantics of the building graph. For example, to be able to check the layout of a sanitary cell for wheelchair-users, or to ensure there are no obstructing elements (like a radiator or fire hose reel) that diminish the passage width. Furthermore, the automation of the SHACL shapes from the building legislation would be a big improvement, since this was done manually in this research. It would also be beneficial to create some form of visual

programming SHACL shapes creator, as proposed in Senthilvel & Beetz [29], so constraints can be customized or additional constraints can be created, depending on the needs of a specific building project.

6. Conclusion

The methodology consisted of three main decisions: using RASE to structure the normative data, using LBD to create the building graph, and using SHACL to define the constraints. While the RASE mark-up proved to be an effective way to manually structure the legislation, it is quite laborious and the efficiency is highly dependent on sentence structures. Representing the building data in an LBD graph was successful since this graph is easy to query and understand. Furthermore, SHACL constraints can easily be defined on the semantic data in an LBD graph, although low-level functions were needed for the implementation, and the scalability of the methodology requires further research.

Acknowledgments

The authors would like to acknowledge the support of the DigiChecks project, which has received funding from the European Union's Horizon Research and Innovation Program under grant agreement no. 101058541 and by the Research Foundation Flanders (FWO), as a Strategic Basic Research grant (grant no. 1S99020N).

References

[1] Inter, Evaluatieonderzoek Vlaamse Toegankelijkheidsverordening, Final Report, Agentschap Toegankelijk Vlaanderen, 2019.
[2] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific american 284 (2001) 34–43.
[3] R. Rudman, R. Bruwer, Defining Web 3.0: opportunities and challenges, The Electronic Library 34 (2016) 132–154.
[4] W3C, Data, 2015. URL: https://www.w3.org/standards/semanticweb/data.
[5] W3C, SPARQL 1.1 Overview, 2013. URL: https://www.w3.org/TR/sparql11-overview/.
[6] W3C, Shapes Constraint Language (SHACL), 2017. URL: https://www.w3.org/TR/shacl/.
[7] M. Bonduel, J. Oraskari, P. Pauwels, M. Vergauwen, R. Klein, The IFC to Linked Building Data Converter - Current Status, in: Proceedings of the 6th Linked Data in Architecture and Construction Workshop, London, UK, 2018, pp. 34–43.
[8] P. Pauwels, A. Roxin, SimpleBIM: From full ifcOWL graphs to simplified building graphs, in: 11th European Conference on Product and Process Modelling, Limasol, Cyprus, 2016, pp. 11–18.
[9] M. H. Rasmussen, M. Lefrançois, G. F. Schneider, P. Pauwels, BOT: The building topology ontology of the W3C linked building data group, Semantic Web 12 (2020) 143–161.
[10] A. Goossens, J. De Smedt, J. Vanthienen, Extracting Decision Model and Notation models from text using deep learning techniques, Expert Systems with Applications 211 (2023).

[11] OMG, Decision Model and Notation, 2021. URL: https://www.omg.org/spec/DMN/1.4/Beta1/PDF.
[12] M. Häußler, S. Esser, A. Borrmann, Code compliance checking of railway designs by integrating BIM, BPMN and DMN, Automation in Construction 121 (2021).
[13] J. Dimyadi, R. Amor, Regulatory knowledge representation for automated compliance audit of BIM-based models, in: 30th International Conference on Applications of IT in the AEC, Beijing, China, 2013, pp. 68–78.
[14] E. Hjelseth, N. Nisbet, Capturing normative constraints by use of the semantic mark-up RASE methodology, in: Proceedings of the CIB W78-W102, Sophia Antipolis, France, 2011, pp. 1–10.
[15] P. Xue, S. Poteet, A. Kao, D. Mott, D. Braines, Constructing Controlled English for Both Human Usage and Machine Processing, International Web Rule Symposium (2013).
[16] A. G. O. S. S.A., nlp.js, 2020. URL: https://github.com/axa-group/nlp.js.
[17] M. H. Rasmussen, Schlachter, LD-BIM, 2023. URL: https://ld-bim.web.app/.
[18] P. Pauwels, S. Zhang, Semantic Rule-checking for Regulation Compliance Checking: An Overview of Strategies and Approaches, in: 32nd CIB W78 Conference, Eindhoven, Netherlands, 2015, pp. 619–628.
[19] J. Werbrouck, M. Senthilvel, J. Beetz, P. Pauwels, A Checking Approach for Distributed Building Data, in: 32. Forum Bauinformatik, Proceedings, Berlin, Germany, 2019, pp. 173–181.
[20] J. Oraskari, M. Senthilvel, J. Beetz, SHACL is for LBD what mvdXML is for IFC, in: The 38th CIB W78 conference on Information and Communication Technologies for AECO, Luxembourg, 2021, pp. 693–702.
[21] D. Elshani, A. Lombardi, A. Fisher, S. Staab, D. Hernández, Inferential Reasoning in Co-Design Using Semantic Web Standards alongside BHoM, in: 33. Forum Bauinformatik, München, Germany, 2022, pp. 89–97.
[22] S. Zentgraf, P. Hagedorn, M. König, Multi-requirements ontology engineering for automated processing of document-based building codes to linked building data properties, IOP Conference Series: Earth and Environmental Science 1101 (2022).
[23] A. T. Kovacs, A. Micsik, BIM quality control based on requirement linked data, International Journal of Architectural Computing 19 (2021) 431–448.
[24] buildingSMART, Information Delivery Specification IDS, 2023. URL: https://technical.buildingsmart.org/projects/information-delivery-specification-ids/.
[25] J. Oraskari, IFCtoLBD, 2023. URL: https://github.com/jyrkioraskari/IFCtoLBD.
[26] A. Sommer, N. Car, pySHACL, 2022. URL: https://github.com/RDFLib/pySHACL.
[27] D. Greenwood, S. Lockley, S. Malsane, J. Matthews, Automated compliance checking using building information models, in: The construction, Building and Real Estate Research Conference of the Royal Institution of Chartered Surveyors, Dauphine Université, Paris, 2010.
[28] A. Hogan, The Web of Data, Springer International Publishing, Cham, 2020.
[29] M. Senthilvel, J. Beetz, A Visual Programming Approach for Validating Linked Building Data, in: EG-ICE 2020 Workshop on Intelligent Computing in Engineering, Berlin, Germany, 2020, pp. 403–411.