Validation of IfcOWL datasets using SHACL

Sander Stolk[1] and Kris McGlinn[2]

[1] Semmtech BV, Amsterdam, the Netherlands
sanderstolk@semmtech.nl
[2] ADAPT Centre, Trinity College, Dublin, Ireland
kris.mcglinn@adaptcentre.ie

Abstract. Standardisation is an important part of ensuring data interoperability. Industry Foundation Classes (IFC) is the current leading standard for BIM in the Architecture Engineering and Construction (AEC) industry and ifcOWL is a Resource Description Framework (RDF) representation of IFC, which enables the interlinking of IFC models with other building and building related data that are also represented using RDF, such as devices, sensor data, geolocation, etc. IFC has a complex schema, designed to support parametric modelling in AEC and adherence to this schema is required to support importing IFC models into popular CAD tools such as Autodesk and ArchiCAD. Therefore, for those wishing to create ifcOWL models which can then be imported into these tools, a process of validation of the output must be done. In this paper, we present a method for validating ifcOWL models using SHACL which can be reused by anyone generating ifcOWL models and which returns a report highlighting any issues identified. The method is tested to validate the outputs of a conversion of geospatial data into IFC using a declarative mapping approach called R2RML.

Keywords: Industry Foundation Classes · SHACL · Linked Data

1 Introduction

Standardisation is an important part of ensuring data interoperability. Industry Foundation Classes (IFC), developed by buildingSMART[3], is the current leading standard for sharing building data in the Architecture Engineering and Construction (AEC) industry. IFC has serializations in STEP, XML and RDF alongside an OWL version of IFC4, called ifcOWL [4], which is built upon Linked Data principles and can therefore support interlinking between other data sets expressed using RDF, over the Web, of which the benefits have been highlighted in numerous research papers [16][12]. As of today, though, the major tool vendors (ArchiCAD [6], Revit [1], etc.) do not support ifcOWL, and still export & import IFC in the XML or STEP file format. Therefore, for those who wish to use ifcOWL with these tools, a process of conversion into one of these formats is required.

[3] www.buildingsmart.org

Approaches exist to convert ifcOWL into STEP [21], but if the initial ifcOWL file does not support the IFC schema correctly, the conversion may not fail, but the tools will not correctly import the file. A challenge for those who wish to use ifcOWL is therefore creating ifcOWL models which adhere correctly to the schema. This is non-trivial as the IFC schema, based on the EXPRESS data modelling language, maintains a complex component hierarchy. For those who wish to generate ifcOWL, a method to support the validation of generated ifcOWL models that can inform them about issues would be beneficial, and provide support to those who are not familiar with IFC's complex schema.

In this paper we present a method for validating ifcOWL using SHACL constraints. SHACL is a standard for validating asserted RDF data, and was published as a W3C Recommendation in 2017 [9]. By expressing constraints using SHACL terminology, one can indicate what information needs to be present in models in order for them to be considered accurate and complete. Tooling that supports this standard, readily available, can afterwards be used to provide a report on the conformance of a model to the captured set of constraints. As a case study, the SHACL constraints are tested on data generated during the conversion of geospatial data into ifcOWL using R2RML rules. The paper details the methodology, the SHACL constraints, and analysis and findings with respect to the approach.

2 Related Work

2.1 Building Information Modelling, Industry Foundation Classes and Linked Data

Building Information Modelling (BIM) is a concept created to support the maintenance and management of data generated across a building's life cycle (BLC) and describes an integrated data model for storing information, typically relating to the functional and physical characteristics of a building [3]. Its primary focus includes a 3D model of the architectural design, detailing positions and dimensions of a building's walls, rooms, windows etc. as well as non-physical building features such as the building costs, accessibility, safety, security and sustainability [17].

The leading standard for BIM are the Industry Foundation Classes[4] (IFC), a STEP-based format for describing buildings in their entirety. The main goal of IFC is to enable open data exchange within and between all available disciplines of the building and construction sector and serve as interoperable standard between the different authoring tools. In theory, this should enable one to export data to an IFC in one application, and build up precisely the same data in the same, or in another application, although there are limits to this with for example the buildings geometry, due to the different implementations of geometric kernels, application logic, and levels of detail [12].

Models described within the IFC format can be serialized in different serialization schemas: STEP-based (.ifc), XML-based (.ifcXML), and RDF-based (.rdf or .ttl). The latter example support the application of Linked Data (LD), an

[4] https://www.buildingsmart.org/about/what-is-openbim/ifc-introduction/

approach to expose, share, and connect related data, which was not previously linked, on the Web [2]. LD makes use of the Resource Description Format (RDF) to represent, store, and link data. RDF expresses data as triples, within a directed graph. Concepts (and individuals) are represented as nodes, and the relationship between these as edges. Using this approach, relationships can be created between data resources linking different domains across the Web. This interlinking of domains has significant potential in the AEC industry, where data related to different domains are generated and consumed across the BLC. Each stage of the BLC, from design to construction and maintenance, requires data sourced from a large set of disparate domains; building geometry and topology data, product data, sensor data, geospatial data, etc.

Currently, no commercialized BIM authoring tools support the use of the RDF-based serialisations of IFC, called ifcOWL. The ifcOWL ontology [15] is based on a conversion procedure that transforms the EXPRESS schema of IFC into an OWL ontology, thereby allowing direct queries and inferences using semantic query languages (SPARQL) and rule languages (e.g., SWRL, N3Logic, SPIN and SHACL). ifcOWL strictly follows the EXPRESS schema of IFC in order to allow bidirectional conversion, resulting in an ontology which is complex and aims to capture the entire building data within one schema. Although efforts were made to split the ifcOWL into modules representing different domains [18], the ifcOWL is still closer related to concepts introduced within EXPRESS, STEP and IFC than to those of the Semantic Web. For those who wish to import their models into BIM authoring tools, it is therefore necessary to convert the resulting ifcOWL file back into STEP.

Methods exist which support the conversion of ifcOWL back into STEP, such as the openly available ifcOWL-to-ifcSTEP converter [21]. Still, when converting there is no checking to ensure the output adheres to IFC in a way that will be understood by the authoring tools. The ifcOWL model must therefore be validated to identify any issues in the model prior to conversion.

SHACL, published as a W3C Recommendation in 2017 [9], is a standard for validating asserted RDF data. By expressing constraints using SHACL terminology, one can indicate what information needs to be present in models in order for them to be considered accurate and complete. Tooling that supports this standard, readily available, can afterwards be used to provide a report on the conformance of a model to the captured set of constraints. Since its publication, SHACL has been adopted in norms for validating exchanged information on the built environment.[5]

2.2 Validation using OWL

Before SHACL was published, data validation was not uncommonly done using terminology from OWL. The Web Ontology Language (or OWL) builds on top of RDF to allow for ontological facts to be expressed. These facts include whether relations between two concepts (or, rather, classes) are transitive,

[5] See, for instance, the Dutch norm NTA8035 for modelling such information on assets. https://www.nen.nl/NEN-Shop-2/Standard/NTA-80352020-nl.htm

intransitive, symmetric or asymmetric. Moreover, OWL offers terminology to capture *restrictions*. This modelling construct allows one to better define a given class by indicating what holds for any of its instances, e.g., all instances of the class Trike have exactly three wheels as parts.

OWL, like the languages it has been built on top of (i.e., RDF and RDFS), was designed for inferencing. As has been pointed out, "OWL restrictions are not actually data constraints, but rather describe inferences to be applied based on them" [8]. In other words, OWL was not meant to be used to validate instances in the same way as one would validate data captured in XML against an XML Schema. Instead, reasoners following the specifications of OWL will interpret property restrictions as axioms to add information to instances that seems to be missing. According to the design principles of OWL, such absent information may simply not yet have been captured or may be captured elsewhere; what is known as the Open World Assumption. Stating that any Trike must have exactly 3 wheels as parts, for instance, indicates that OWL reasoners may infer the existence of 3 wheels for any instance of a Trike where they have not yet been mentioned and to add that information to the dataset at hand.

Even though OWL is intended to be used for inferencing, it has not been uncommon to see it used for data validation purposes instead. The modelling construct of OWL property restrictions lent itself well to express data constraints, for which similar information tends to be captured (e.g., minimum cardinality, maximum cardinality, value ranges) [8]. In effect, the appropriation applies a Closed World Assumption to validate the data at hand. Indeed, this is the approach that has been taken with ifcOWL. The risk with such an approach, however, is that information expressed in this manner can be interpreted incorrectly by applications as its use deviates from the standard. The consortium behind OWL, W3C, has therefore worked on an alternative precisely for expressing data validation constraints: SHACL.

2.3 Validation using SHACL

The Shapes Constraint Language (or SHACL) was published as W3C recommendation in 2017 [9]. The vocabulary allows one to model data constraints by means of so-called shapes. A shape consists of a set of restrictions placed on certain data elements, such as classes. In such a case, all instances of that class will be validated using the set of restrictions described by the shape. As mentioned earlier, such restrictions on classes are currently expressed in ifcOWL using OWL terminology where, instead, SHACL would be more appropriate.

The incorrect use of OWL for modelling data validation constraints is one acknowledged by the editors of SHACL. In fact, there is ample information available on what it means to migrate from OWL to SHACL for data constraints. Holger Knublauch, one of the editors of SHACL, has written a comparison of SHACL and OWL [8]. This comparison includes a table that indicates counterpart terminology for OWL and SHACL, which facilitates migrating data validation from the former to the latter. Using counterpart terminology, it is therefore possible to capture the data constraints present in ifcOWL using SHACL instead of OWL. Doing so enables the leveraging of readily-available

SHACL tooling[6] for validating IFC models for conformance to the STEP file format and associated EXPRESS schema.

3 Methodology & Results

3.1 Obtaining SHACL Shapes for ifcOWL

As mentioned, ifcOWL currently captures its data validation constraints using terminology from OWL. To capture that same information using counterpart SHACL terminology instead, this paper explores two methods.

Method 1: Applying rules to ifcOWL. The first method in obtaining SHACL for ifcOWL involves two steps: (1) extracting the data validation information from ifcOWL and (2) transforming it to appropriate SHACL data. For this approach, we employ querying and transformation mechanisms available for RDF and apply these to ifcOWL.

The mechanisms used in this paper for this method are SPARQL and SHACL Advanced Features [19][20]. These allow one to model, in RDF, queries (using the standard querying language SPARQL) and rules that infer new information based on the selection (which are expressed using SHACL Advanced Features). Thus, the process can rely on an explicit, RDF model and be performed in an automated manner using open-source tooling that supports these features.[7] The explicit model effectively acts as input that specifies what needs to be selected and how it ought to be transformed. This input model, created for this paper, is shown in Listing 1.3 and will henceforth be referred to as the *rules model*.

The rules model contains three main elements. The first is an ontology resource which, most notably, is used to declare prefixes used in SHACL rules. The second element is a SHACL rule which will be executed for every instance of an owl:Class. This rule contains a query that constructs SHACL data based on its selection of OWL property restrictions that apply to the Class instance. The WHERE-clause of the query retrieves every possible pattern of OWL property restrictions in ifcOWL and combines these through UNIONs before the SHACL data is constructed based on this information. The third element in the rules model is a function called *getIfcClassForShape*, which retrieves valid value types for properties.[8]

The rules model is executed on an ifcOWL model using one of the available, open-source SHACL applications. By providing the application with ifcOWL (in this paper, ifcOWL version IFC2X3_TC1) as its 'datafile' and the rules model as its 'shapesfile', the application fetches the required information and constructs the desired SHACL data corresponding to the data validation restrictions in the original ifcOWL. Listing 1.1 contains an example snippet showing the results of the transformation for the ifcOWL class IfcDoorPanelProperties.

[6] See, for instance, TopBraid SHACL API (https://github.com/TopQuadrant/shacl) and pySHACL (https://github.com/RDFLib/pySHACL).
[7] See the previous footnote.
[8] This function is elaborated on after the description of both methods to obtain SHACL shapes.

Listing 1.1: ifcOWL data constraints in OWL (left) and SHACL (right)

```
ifc:IfcDoorPanelProperties
  rdf:type owl:Class ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom ifc:IfcPositiveLengthMeasure ;
      owl:onProperty ifc:panelDepth_IfcDoorPanelProperties
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
      owl:onProperty ifc:panelDepth_IfcDoorPanelProperties ;
      owl:onClass ifc:IfcPositiveLengthMeasure
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom ifc:IfcDoorPanelOperationEnum ;
      owl:onProperty ifc:panelOperation_IfcDoorPanelProperties
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
      owl:onProperty ifc:panelOperation_IfcDoorPanelProperties ;
      owl:onClass ifc:IfcDoorPanelOperationEnum
    ]
  .
```

```
ifc:IfcDoorPanelProperties
  rdf:type sh:NodeShape ;
  sh:property
    [
      sh:class ifc:IfcPositiveLengthMeasure ;
      sh:path ifc:panelDepth_IfcDoorPanelProperties
    ] ;
  sh:property
    [
      sh:qualifiedMaxCount 1 ;
      sh:path ifc:panelDepth_IfcDoorPanelProperties ;
      sh:qualifiedValueShape
        [
          sh:class ifc:IfcPositiveLengthMeasure
        ]
    ] ;
  sh:property
    [
      sh:class ifc:IfcDoorPanelOperationEnum ;
      sh:path ifc:panelOperation_IfcDoorPanelProperties
    ] ;
  sh:property
    [
      sh:qualifiedMinCount 1 ;
      sh:qualifiedMaxCount 1 ;
      sh:path ifc:panelOperation_IfcDoorPanelProperties ;
      sh:qualifiedValueShape
        [
          sh:class ifc:IfcDoorPanelOperationEnum
        ]
    ]
  .
```

This approach has two main advantages compared to the second method. Firstly, it is easy to set up as it reuses SHACL mechanisms. The SHACL tooling used in this paper, for instance, contains functionality both for running data validations and for executing SHACL rules. The second advantage is that information of ifcOWL, and all relations between them, have already been linked (in an RDF form) and can be queried easily.

Method 2: Incorporating into ifcOWL generation. The second method explored in this paper to obtain SHACL constraints for ifcOWL is to make it part of the process in which ifcOWL itself is formed. The tool with which ifcOWL is generated from standardized EXPRESS schemas (used with the STEP file format) is done with a tool called EXPRESStoOWL [15]. The functionality of this tool can be extended so that the ifcOWL data validation restrictions that it outputs are expressed using SHACL instead of OWL. We found that such alterations are best captured in a new Writer class added to the code: a SHACLWriter.java next to the existing OWLWriter.java. Listing 1.4 contains snippets of both these Java classes, in which OWL terminology used for data validation in OWLWriter.java has been replaced by SHACL terminology in SHACLWriter.java.

This second method of obtaining SHACL would allow incorporating SHACL in the very creation process of ifcOWL. If, in the future, OWL restrictions were indeed discarded for newer iterations of ifcOWL (when favouring SHACL, for instance), this method will still be capable of producing appropriate data constraints expressed with SHACL. Unfortunately, implementing this method presents a greater challenge to implement than the first-mentioned method and has not yet been completed at the time of writing (and has therefore not been tested fully). The next section will elaborate on this matter.

Basic datatypes and data constraints. IfcOWL contains a number of properties that can be used in models. These properties should always have values of a certain type. The property *length value*, for instance, can be said to always need values of type *IfcLengthMeasure*. This restriction, taken from ifcOWL version IFC2x3_TC1, is shown in the RDF snippet below.

```
[ a  owl:Restriction
  owl:onProperty     ifc:lengthValue_IfcQuantityLength ;
  owl:allValuesFrom  ifc:IfcLengthMeasure ]
```

In effect, as OWL terminology is used to capture this restriction, it ought to be read as an inference axiom: Use of the property for length value, regardless of whether it leads to a valid value or not, will be inferred to (also) be of type *IfcLengthMeasure*. Such inferred information may be desired if an ifcOWL model with instance data is valid and correct, but it would be misleading when applied to invalid values – such as when a date is asserted as length value.

With SHACL it is possible to express the information as data constraint instead. IfcOWL models then ought to have a value typed as *IfcLengthMeasure* when asserting a *length value*. An RDF snippet to that extent is shown below. It could be further improved, however, by taking into account current practices.

```
[ sh:path   ifc:lengthValue_IfcQuantityLength ;
  sh:class  ifc:IfcLengthMeasure ]
```

Tooling available to transform IFC data from a STEP file to an ifcOWL model, types values of the property length value as *express:REAL* rather than *IfcLengthMeasure* [21]. Doing so may be a pragmatic choice. The last-mentioned type is, in fact, a specialization of the former. Capturing a value on the more generic level (i.e., *express:REAL*) allows the reuse of such values for multiple purposes instead of only as value to the property length value. For practical purposes, the value itself (e.g., "12.4") will remain unchanged. As a consequence, it begs the question whether data validation of ifcOWL models should treat captured real numbers for a length value as invalid when that value is not expressed at the most specific level (i.e., *IfcLengthMeasure*) but at the most generic level (i.e., *express:REAL*). In both situations, it would still be possible to convert the information to a correct STEP file that can be read by CAD tooling.

In order to consider both specific and generic levels of typing values in an ifcOWL model correct, the SHACL restriction regarding relevant properties ought to require (at the minimum) the most generic datatype available. To illustrate, the case described thus far would then be captured in SHACL as follows:

```
[ sh:path   ifc:lengthValue_IfcQuantityLength ;
  sh:class  express:REAL ]
```

This stance has already been incorporated in the first method described in this paper. In fact, the function *getIfcClassForShape* in the rules model takes care of exactly this selection. If a more generic datatype (i.e., stemming from

EXPRESS) can be used in a constraint for a property value, it is preferred over the most specific one.[9] Implementing this stance in the first method was relatively easy, since the hierarchy between different datatypes can be queried. Implementing it in the second method, however, will take more effort. There, this hierarchy is still to be made accessible in code before it can be used in writing the desired SHACL output. The next section will demonstrate how models can be validated once SHACL shapes are available (regardless of which method has been used to create them).

3.2 Validating IFC models using SHACL

Once SHACL shapes have been obtained that express the data constraints for IFC models (obtained for this article by means of the first-mentioned method), data validation can be done with any SHACL-compliant tool. SHACL compliant tooling validates whether instances of ifcOWL classes in the IFC model conform to the constraints placed on them. The resulting output is a validation report in RDF, indicating whether the entire IFC model conforms to the constraints and, if it does not conform, which violations are present. For this paper, we have used the same open-source tool that has been utilized for generating SHACL data in 'Method 1' above.[10] Listing 1.2 contains an RDF snippet of a validation report created for an IFC model that was generated based on geospatial data.

Listing 1.2: RDF snippet that contains a SHACL data validation report.

```
@prefix ifc:   <http://www.buildingsmart-tech.org/ifcOWL/IFC2X3_TC1#> .
@prefix sh:    <http://www.w3.org/ns/shacl#> .
@prefix data:  <http://data.geohive.ie/resource/ifcbuilding/> .

[ a sh:ValidationReport ;
  sh:conforms
    false ;
  sh:result
    [ a sh:ValidationResult ;
      sh:focusNode
        data:IfcGloballyUniqueId_361c76fd-cdd3-4c56-aae1-848f3baecf77IfcOwnerHistory ;
      sh:resultMessage
        "Value must be an instance of ifc:IfcChangeActionEnum" ;
      sh:resultPath
        ifc:changeAction_IfcOwnerHistory ;
      sh:resultSeverity
        sh:Violation ;
      sh:sourceConstraintComponent
        sh:ClassConstraintComponent ;
      sh:sourceShape
        []  ;
      sh:value
        <http:ifcowl.openbimstandards.orgIFC2X3_TC1#ADDED>
    ] ;
  sh:result
    [ a sh:ValidationResult ;
      sh:focusNode
        data:IfcGloballyUniqueId_361c76fd-cdd3-4c56-aae1-848f3baecf772IfcWallStandardCase ;
      sh:resultMessage
        "Value must be an instance of ifc:IfcOwnerHistory" ;
```

[9] The only exception are enumerations, since validation will require to know which exact kind of enumeration is necessary for a certain property.

[10] https://github.com/TopQuadrant/shacl

```
31          sh:resultPath
32            ifc:ownerHistory_IfcRoot ;
33          sh:resultSeverity
34            sh:Violation ;
35          sh:sourceConstraintComponent
36            sh:ClassConstraintComponent ;
37          sh:sourceShape
38            _:b2 ;
39          sh:value
40            data:IfcGloballyUniqueId_361c76fd-cdd3-4c56-aae1-848f3baecf772IfcOwnerHistory
41          ] ;
42      .
```

The validation report in Listing 1.2 indicates that the IFC model provided does not conform to the SHACL shapes for ifcOWL. Two results can be seen that are violations (see lines 17 and 34). The first violation in the IFC model is a resource with a property *changeAction* that does not contain a valid value. The value required ought to be one out of an enumeration, but instead a flawed value was present (see line 23; forward slashes required in the URI value are absent). The second validation result shown in this report indicates that a value of the property *ownerHistory* has not been typed, as is mandatory, as an instance of the IFC class *IfcOwnerHistory*. Thus, it is possible to obtain a list of these and other violations in order to make corrections to the IFC model – or, in this case, the conversion process that generates the model from available geospatial data – and ensure that the IFC model can afterwards be transformed to the STEP format successfully and read by existing IFC software that relies on valid STEP input.

4 Validation of sample ifcOWL model

To further test the SHACL constraints, a sample ifcOWL model was used which was generated as part of ongoing work to convert geospatial data directly into ifcOWL using the Relational Database to Resource Description Framework Language (R2RML)[5], a standardised declarative mapping language which allows one to convert non-RDF resources into RDF while relying on the underlying relational database technology (usually SQL) to manipulate the data. R2RML enables the user to create mappings which include target vocabularies, so that for example, columns in a table can be assigned specific definitions given by existing vocabularies on the web. This is a powerful tool that enables the re-use of common vocabularies to describe data sets and for bringing semantics to tabular data through conversion to RDF.

In parallel work (under peer review), R2RML has been used to convert subsets of the Ordnance Survey Ireland (OSi, Ireland's national mapping agency) Prime2 data set (information of over 50 million spatial objects including road segments, buildings, fences, etc.) using R2RML [7][11][13][12]. Recent work has seen the publication of over 200 thousand buildings (polygon foot print, geodetic coordinate), being made available as RDF using the GeoSPARQL vocabulary [10][11]. This data meets some basic requirements for BIM. R2RML has been demonstrated to provide a method for generating 3D semantic BIM

[11] http://data.geohive.ie/downloadAndQuery.html

from relatively simple geospatial inputs (a 2D footprint), which is an important step towards making BIM models open and available to support a wide range of use cases. The mappings and resulting ifcOWL model can be found here: [12].

A major challenge when converting into ifcOWL is related to the complexity of the IFC schema and its RDF representation as ifcOWL. ifcOWL maintains a complex set of relationships which includes, for example, lists to represent nested placement of entities within a building along with their orientation, and lists to represent each dimension of a coordinate as a single value [14], and lists to represent geolocation [12]. Each list requires its own mapping within the R2RML file. IFC also includes a lot of metadata with each model, such as the different units used, the creator of the file and the application used to generate it. The R2RML mapping for this purpose should therefore correctly model all the relationships required.

If these are not modelled correctly, the generated ifcOWL will be incorrect and if, for example, one wishes to convert the ifcOWL back into STEP so that it can be imported into a CAD tool, using the openly available ifcOWL-to-ifcSTEP converter [21], the generated IFC STEP will fail on import. SHACL therefore provides an important step in the validation of the ifcOWL and the R2RML, by ensuring that the generated ifcOWL is valid. The validation reports that, using the aforementioned method, have been generated have assisted in fine tuning the R2RML transformation to ensure the model's conformance and completeness for processing in BIM CAD tools.

5 Conclusion & Future Work

This paper has demonstrated the process for data validation of ifcOWL models by means of SHACL. Unlike OWL, which was designed for inferencing, SHACL was designed specifically for data validation. By migrating ifcOWL data constraints expressed as OWL restrictions to SHACL shapes, ifcOWL models can benefit from the use of standardized and readily-available data validation tooling. The validation reports generated by SHACL tools highlight issues identified in a model. A sample of such a report has been included in this paper for an ifcOWL model created through a conversion of geospatial data.

Performing data validations is an important part in ensuring data interoperability. The described data validation process using SHACL allows one to identify mistakes in a model and make corrections before processing it further – such as importing the model into popular CAD tools. Future versions of ifcOWL, then, would benefit from incorporating SHACL shapes. Steps to gradually adopt SHACL for data validation, and incorporating it into the very generation process of the ifcOWL standard, are therefore recommended.

References

1. Autodesk: Revit (2017), https://www.autodesk.eu/products/revit
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems **5**(3), 1–22 (jul 2009)

[12] https://www.scss.tcd.ie/~mcglink/r2rml/

3. Borrmann, A., König, M., Koch, C., Beetz, J.: Building Information Modeling: Technology Foundations and Industry Practice. Springer International Publishing (2018), ISBN: 978-3-319-92861-6
4. buildingSMART International Ltd.: ifcOWL - buildingSMART Technical (2019), https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/
5. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C Recommendation (September 2012), 1–34 (2012), https://www.w3.org/TR/r2rml/
6. Graphisoft: ARCHICAD (2016), https://www.graphisoft.com/archicad/
7. Ireland, O.S.: GeoHIVE - Bringing Irish Geospatial Data to the Web (2019), http://data.geohive.ie/
8. Knublauch, H.: SHACL and OWL Compared (2017), https://spinrdf.org/shacl-and-owl.html
9. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL) (August 2016), 1–88 (2017), https://www.w3.org/TR/shacl/
10. Lopez, X.: GeoSPARQL - A geographic query language for RDF data A proposal for an OGC Draft Candidate Standard p. 13 (2012), https://www.opengeospatial.org/standards/geosparql
11. McGlinn, K., Debruyne, C., McNerney, L., O'Sullivan, D.: Integrating Ireland's Geospatial Information to Provide Authoritative Building Information Models. In: Proceedings of the 13th International Conference on Semantic Systems - Semantics2017. vol. 13, pp. 57–64. ACM Press (2017)
12. McGlinn, K., Wagner, A., Pauwels, P., Bonsma, P., Kelly, P., O'Sullivan, D.: Interlinking geospatial and building geometry with existing and developing standards on the web. Automation in Construction pp. 235–250
13. O'Donovan, J., O'Sullivan, D., McGlinn, K.: A method for converting IFC geometric data into GeoSPARQL. In: CEUR Workshop Proceedings. vol. 2389, pp. 7–20 (2019)
14. Pauwels, P., Krijnen, T., Terkaj, W., Beetz, J.: Enhancing the ifcOWL ontology with an alternative representation for geometric data. Automation in Construction **80**, 77–94 (2017)
15. Pauwels, P., Terkaj, W.: EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. Automation in Construction **63**, 100–133 (2016)
16. Pauwels, P., Zhang, S., Lee, Y.C.: Semantic web technologies in AEC industry: A literature overview (jan 2017). https://doi.org/10.1016/j.autcon.2016.10.003
17. Taylor, J., Bernstein, P.: Paradigm Trajectories of Building Information Modeling Practice in Project Networks. Journal of Management in Engineering - J MANAGE ENG **25** (2009)
18. Terkaj, W., Pauwels, P.: A method to generate a modular ifcOWL ontology. In: Borgo, S., Kutz, O., Loebe, F., Neuhaus, F. (eds.) Proceedings of the 8th International Workshop on Formal Ontologies meet Industry. CEUR Workshop Proceedings, vol. 2050. Bolzano, Italy (2017)
19. W3C: SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/ (2013), accessed: 28-Mar-2020
20. W3C: SHACL Advanced Features. https://www.w3.org/TR/shacl-af/ (2017), accessed: 28-Mar-2020
21. Zhang, B.: Roundtrip converters from IFC STEP files to IfcOWL RDF files, https://github.com/BenzclyZhang/IfcSTEP-to-IfcOWL-converters

Listing 1.3: RDF that captures SHACL rules for obtaining SHACL from ifcOWL.

```
@prefix :     <http://example.org/rules/ifcowl-to-shacl/> .
@prefix sh:   <http://www.w3.org/ns/shacl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix list: <https://w3id.org/list#> .
@prefix expr: <https://w3id.org/express#> .
@prefix dct:  <http://purl.org/dc/terms/> .

:  a  owl:Ontology ;
    rdfs:label "Rules: IfcOWL to SHACL"@en ;
    rdfs:comment "SHACL-AF rules to infer SHACL shapes from ifcOWL."@en ;
    dct:creator <https://www.universiteitleiden.nl/en/staffmembers/sander-stolk> ;
    owl:imports sh: ;
    sh:declare
      [ sh:prefix "" ;
        sh:namespace "http://example.org/rules/ifcowl-to-shacl/"^^xsd:anyURI ; ],
      [ sh:prefix "sh" ;
        sh:namespace "http://www.w3.org/ns/shacl#"^^xsd:anyURI ; ] ,
      [ sh:prefix "rdf" ;
        sh:namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns#"^^xsd:anyURI ; ],
      [ sh:prefix "rdfs" ;
        sh:namespace "http://www.w3.org/2000/01/rdf-schema#"^^xsd:anyURI ; ],
      [ sh:prefix "owl" ;
        sh:namespace "http://www.w3.org/2002/07/owl#"^^xsd:anyURI ; ],
      [ sh:prefix "xsd" ;
        sh:namespace "http://www.w3.org/2001/XMLSchema#"^^xsd:anyURI ; ] .


owl:Class  a  sh:NodeShape ;
    sh:rule [
        a sh:SPARQLRule ;
        sh:prefixes : ;
        sh:construct """
CONSTRUCT {
    $this a rdfs:Class .
    $this a sh:NodeShape .
    $this sh:property ?tgtPropertyShape .
    ?tgtPropertyShape sh:path ?onProperty .
    ?tgtPropertyShape sh:datatype ?datatype .
    ?tgtPropertyShape sh:class ?shacl_class .
    ?tgtPropertyShape sh:in ?enumList .
    ?tgtPropertyShape sh:minCount ?shacl_minC .
    ?tgtPropertyShape sh:maxCount ?shacl_maxC .
    ?tgtPropertyShape sh:qualifiedValueShape ?shacl_qVS .
    ?tgtPropertyShape sh:qualifiedMinCount ?shacl_minQC .
    ?tgtPropertyShape sh:qualifiedMaxCount ?shacl_maxQC .
    ?tgtPropertyShape sh:hasValue ?hasV .
}
WHERE {
    $this rdf:type/(rdfs:subClassOf)* owl:Class .
    OPTIONAL {
        $this rdfs:subClassOf ?restriction .
        ?restriction owl:onProperty ?onProperty .
        OPTIONAL {
            ?onProperty rdfs:range ?range .
            BIND ((STRSTARTS(str(?range), str(xsd:)) || (?range = rdf:langString)) AS
↪   ?isDatatype) .
            BIND (IF(  ?isDatatype, ?range, ?null) AS ?datatype) .
            BIND (IF(! ?isDatatype, ?range, ?null) AS ?nonDatatype) .
            OPTIONAL {
                ?range owl:oneOf ?enumList .
            } .
            BIND (IF((!bound(?enumList)), ?nonDatatype, ?null) AS ?rangeClass) .
            BIND (IF(bound(?rangeClass), :getIfcClassForShape(?rangeClass), ?null) AS
↪  ?shacl_class) .
        } .
        {
            ?restriction owl:cardinality ?c .
            BIND (?c AS ?minC) .
            BIND (?c AS ?maxC) .
        } UNION {
            ?restriction owl:minCardinality ?minC .
        } UNION {
            ?restriction owl:maxCardinality ?maxC .
        } UNION {
            ?restriction owl:minQualifiedCardinality ?minQC .
            ?restriction owl:onClass ?onClass .
            BIND (:getIfcClassForShape(?onClass) AS ?shacl_qVS) .
        } UNION {
            ?restriction owl:maxQualifiedCardinality ?maxQC .
            ?restriction owl:onClass ?onClass .
            BIND (:getIfcClassForShape(?onClass) AS ?shacl_qVS) .
        } UNION {
            ?restriction owl:qualifiedCardinality ?qC .
            ?restriction owl:onClass ?onClass .
            BIND (?qC AS ?minQC) .
            BIND (?qC AS ?maxQC) .
            BIND (:getIfcClassForShape(?onClass) AS ?shacl_qVS) .
        } UNION {
            ?restriction owl:allValuesFrom ?allVF .
            BIND (:getIfcClassForShape(?allVF) AS ?shacl_class) .
        } UNION {
            ?restriction owl:someValuesFrom ?someVF .
            BIND (1 AS ?minQC) .
            BIND (:getIfcClassForShape(?someVF) AS ?shacl_qVS) .
        } UNION {
            ?restriction owl:hasValue ?hasV .
        } .
        BIND (BNODE() AS ?tgtPropertyShape) .
        BIND (STRDT(?minC, xsd:integer) AS ?shacl_minC) .
        BIND (STRDT(?maxC, xsd:integer) AS ?shacl_maxC) .
        BIND (STRDT(?minQC, xsd:integer) AS ?shacl_minQC) .
        BIND (STRDT(?maxQC, xsd:integer) AS ?shacl_maxQC) .
    } .
}""" ; ] .


:getIfcClassForShape
    a sh:SPARQLFunction ;
    rdfs:comment "Returns the EXPRESS superclass of $op1, if it exists, otherwise $op1
↪  itself." ;
    sh:parameter [
        sh:path :op1 ;
        sh:description "The node" ;
    ] ;
    sh:prefixes : ;
    sh:select """
SELECT ?result
WHERE {
    $op1 a ?type .
    OPTIONAL {
        $op1 rdfs:subClassOf* ?superClass .
        FILTER (STRSTARTS(STR(?superClass), "https://w3id.org/express#")) .
        FILTER (?superClass != <https://w3id.org/express#ENUMERATION>) .
    }
    BIND (COALESCE(?superClass, $op1) AS ?result) .
}""" .
```

Listing 1.4: EXPRESStoOWL's OWLWriter.java (top) and new SHACLWriter.java (bottom)

```java
    private void outputOWLproperty(BufferedWriter out, PropertyVO property) {
        try {
            if (property.isList() || property.isArray()) {
                out.write("ifc:" + property.getLowerCaseName() + "\r\n");
                out.write("\trdfs:label \"" + property.getOriginalName()//
                ↪  getOriginalNameLowerCase()
                        + "\" ;\r\n");
                out.write("\trdfs:domain ifc:" + property.getDomain().getName() + "
                ↪  ;\r\n");
```

...

```java
                if (!property.getRangeNS().equalsIgnoreCase("expr")) {
                    // write List range if necessary
                    if (!property.isSet()) {
                        if (property.isListOfList()) {
                            if (!listPropertiesOutput.contains(property.getRange() +
                            ↪  "_List")) {
                                // property not already contained in resulting
                                // OWL file
                                // (.TTL) -> no need to write additional
                                // property

                                listPropertiesOutput.add(property.getRange() + "_List");

                                out.write(property.getRangeNS() + ":" +
                                ↪  property.getRange() + "_List_EmptyList" + "\r\n");
                                out.write("\trdf:type owl:Class ;" + "\r\n");
                                out.write("\trdfs:subClassOf list:EmptyList, " +
                                ↪  property.getRangeNS() + ":" + property.getRange() +
                                ↪  "_List_List" + " ." + "\r\n" + "\r\n");
```

```java
    private void outputSHACLproperty(BufferedWriter out, PropertyVO property) {
        try {
            if (property.isList() || property.isArray()) {
                if (!property.getRangeNS().equalsIgnoreCase("expr")) {
                    // write List range if necessary
                    if (!property.isSet()) {
                        if (property.isListOfList()) {
                            if (!listPropertiesOutput.contains(property.getRange() +
                            ↪  "_List")) {
                                // property not already contained in resulting
                                // OWL file
                                // (.TTL) -> no need to write additional
                                // property

                                listPropertiesOutput.add(property.getRange() + "_List");

                                out.write(property.getRangeNS() + ":" +
                                ↪  property.getRange() + "_List_List" + "\r\n");
                                out.write("\trdf:type sh:NodeShape ;" + "\r\n");

                                out.write("\tsh:property" + "\r\n");
                                out.write("\t\t[" + "\r\n");
                                out.write("\t\t\tsh:path list:hasContents ;" + "\r\n");
                                out.write("\t\t\tsh:class " + property.getRangeNS() + ":"
                                ↪  + property.getRange() + "_List" + "\r\n");
                                out.write("\t\t] ;" + "\r\n");
```