A Visual Programming Approach for Validating Linked Building Data

Madhumitha Senthilvel, Jakob Beetz
Design Computation, Faculty of Architecture, RWTH Aachen University, Germany
senthilvel@caad.arch.rwth-aachen.de

Abstract. The upsurge in the development and availability of specialized BIM-based tools have enabled parsing of information from models into different formats for specific use cases. However, interoperability issues in IFC result in information loss during the conversion between formats, standards, and software. While Semantic Web technologies and Linked Data are promising approaches for representing and linking information spread across sources and formats, the problem of checking data consistency in the tool-chains remains. Approaches such as SPARQL, mvdXML are considered either too verbose or unable to handle non-IFC data. Since SHACL was only recently introduced (2017), visual interfaces for the creation of these rules are not investigated in the AEC domain. Current implementations of SHACL focus on the generation of validation reports and not on the creation of SHACL constraints themselves, which requires both Semantic Web knowledge and domain knowledge. This paper proposes a visual programming interface for creating SHACL shapes to improve the ease of creation and editing of constraints for non-semantic web experts with its focus on supporting AEC ontologies and use-cases. To aid that, this paper explores how constraints can be modularized in SHACL so that they are re-usable across use-cases. The proof of concept is demonstrated using a linked building data example. This work is an initial step towards connecting services, wherein, constraints can be created with minimal domain and expert knowledge, and these constraints facilitate checking the validity of information.

1. Introduction

Building Information Modelling (BIM) is being increasingly accepted as a valuable asset for the construction industry, in terms of design, planning, collaboration and constraint checking. With an upsurge in the development and availability of specialized BIM-based tools, it is now possible to parse information from models into different formats for specific use cases. However, due to interoperability issues in IFC (Industry Foundation Classes) and the checking tools that support it, information is lost during the conversion between formats, standards, and software.

Semantic Web technologies and Linked Data have been identified as promising approaches for representing and linking information spread across sources and formats (Beetz et al., 2009; Pauwels et al., 2011). Nonetheless, the problem of checking data consistency remains. Information still has to be checked for its consistency before being transferred between applications.

Numerous checking approaches such as XML, SPARQL Protocol and RDF Query Language (SPARQL), Shape Expressions (SHeX), Object Constraint Language(OCL), Shape Constraint Language(SHACL), its predecessor SPIN etc. have been employed to validate data. In the AEC domain, prevalent checking languages/standards are mvdXML and SPARQL. mvdXML is widely regarded as the go-to for constraint checking (Chipman et al., 2016). However, its support is limited for IFC based models, with little scope for checking other kinds of data such as photographs, textual information etc. Furthermore, existing implementations of mvdXML, SPARQL require both semantic and domain knowledge for modelling constraints; other modelling approaches such as SHACL, SHeX, OCL do not have implementations yet for AEC domain. The motivation for this research stems from this gap. This paper proposes and explores the implementation of an interface for creating constraints in SHACL using open-source API. The proposed approach focuses on a visual programming interface for creating SHACL shapes to improve the ease of creation and editing of constraints for non-semantic experts, to support AEC ontologies and use-cases.

To gain an understanding of the existing implementations of checking approaches, the next section focuses on mvdXML, and BIMSPARQL- the two most prevalent checking mechanisms in the AEC domain.

2. Existing methods for data validation: approaches, and implementations

The existing approaches for the creation of rules for data validation can be roughly categorized into proprietary methods and non-proprietary methods. Solibri Model Checker (SMC) is a widely-used checking tool for applications such as code compliance and clash detection falls under the Programming based approach (Zhang et al., 2015). While it is based on hard-coding rules, it is also customizable using the in-built Rule Manager option. However, proprietary definitions are employed to implement these rules. This reduces the flexibility of the tool for checking information beyond those envisioned by the tool developers. Such a situation is valid, especially in a linked data environment where non-IFC data would also have to be validated. Non-proprietary model checkers (for example, the mvdXML Checker) overcome vendor-lock-in.

This section begins with discussing the major non-proprietary implementations for data validation in AEC: IFCDoc, mvdXML generator for mvdXML and BIMSPARQL for SPARQL for creating constraints and contrasting them with SHACL. The section aims to give an overview of implementations of the above approaches and the challenges associated with using them. These challenges serve as a prologue to understanding the desired features of an interface for creating constraints for AEC use cases.

2.1 mvdXML

mvdXML is a standard introduced by buildingSMART. It is an electronic format representing the Model View Definitions (MVD), which themselves represent the information required during data exchanges. They are a subset of the data schema and can be obtained from Information Deliver Manual (IDM) and Exchange Requirement (ER). mvdXML documentation describes its application to IFC data schema only (Chipman et al., 2016).

However, mvdXML has some drawbacks, most of which stem from the complex nature of IFC itself. For example, it lacks logical formalisms, it only considers IFC schema, and MVD-based view constructors are not flexible and dynamic (Roxin, 2016). However, implementations for generating rules and checking them have been developed.

The mvdXML Checker is an implementation developed by Zhang et. al which checks for IFC data conformation which uses mvdXML rulesets (Zhang et al., 2015). To function it needs: a mvdXML generator for creating rulesets (*IFCDoc*), the checker itself which checks the generated rulesets against given IFC data, and lastly an output viewer which shows the validation in BCF (Building Collaboration Format). For brevity, *IFCDoc* will be briefly discussed. *IFCDoc* is a tool which creates XML rulesets preloads all the IFC schema releases. It enables checking the existence of a value/entity/attribute, whether it is present in the correct entity type and subtype and finally, the accuracy of the attribute value and cardinality of the said attribute. van Strien gives a comprehensive practical guide to *IFCDoc* (van Strien, 2015).

However, it requires domain end-user to have knowledge of IFC, mvdXML and *IFCDoc* (Weerink, 2016). Currently, *IFCDoc* tool has limited support since it is not being updated. Consequently, the mvdXML generator and checker was hence developed by implementing a user-interface for generating the XML rulesets using spreadsheet-based requirement documentation along with Zhang et. al's original Checker (Weerink, 2016). This generator works specifically for a set of specifications and hence is not generic to accommodate other use-cases.

2.2 SPARQL

SPARQL, which has been implemented for querying and validation in BIMSPARQL, is regarded as complex and has a high threshold for learning due to its verbosity and flexibility to define a constraint in multiple ways (Zhang et al., 2018). However, SPARQL's inherent flexibility for querying a query in multiple ways demands that the user be well aware of the syntax of the language. Additionally, current implementations of SPARQL still necessitate the user to enter queries according to SPARQL syntax, with no masking of the complexities of the language.

3. SHACL as an alternative data validation language

In section 2, existing implementations for checking information were discussed. As of now, only *IFCDoc* exists to generate mvdXML rulesets. While there are limited open-source options for editors/tools to check AEC information, there exists an even more dire shortage of tools for generating the rules which are necessary for the checking process. These tools will need to be able to cater to checking non-IFC data as well. SHACL (Shape Constraint Language), a domain-agnostic constraint language is still not yet investigated for the AEC domain, partially due to it being a recent development[1]. SHACL uses the concept of shapes graphs (rulesets) to define constraints. When a given input (termed as data graph) is validated against shape graphs, a validation report is generated containing the classes violating the rules. SHACL is considered a more general-purpose validation language since it can be used for any information encoded in the JSON/turtle format.

At present, SHACL has a few implementations such as *TopBraid's SHACL API*[2], *SHACL Playground*[3], *pySHACL*[4] and *unSHACLed*[5] etc. The former 4 focus on the validation of data against the shapes, and not on the creation of the constraints (shapes) themselves. Only *unSHACLed* implements a prototype based on an interface for a drag-and-drop option for creation of SHACL shapes (Meester et al., 2019). This feature of drag and drop helps mask the complexity of the language and make it easy for all levels of users to utilize the tool.

However, in the AEC domain, the challenge in integrating SHACL in practice with other tools is that such constraint definition languages require not only expert semantic web knowledge but also knowledge on assessing the applicability of these rules for AEC ontologies and use-cases. In a visual programming approach, the syntax of SHACL can be masked by the code blocks. General purpose visual programming languages have been shown to be more user-friendly and accessible for non-domain users (Catarci and Santucci, 1995).

The partially pre-programmed editable code-blocks (called nodes) can be connected non-linearly, hence facilitating the quicker generation of SHACL shapes. In this approach, the code blocks contain the reusable SHACL shapes (constraints) and the associating data object against

[1] Introduced as a standard in 2017.
[2] https://github.com/TopQuadrant/shacl
[3] https://shacl.org/playground/
[4] https://pypi.org/project/pyshacl/
[5] http://ecodalo.ilabt.imec.be:8980/#/

which it is being validated. Thus the user only has to determine the type of constraints (such as cardinality, data type, range etc.), and the associated class and property for which these constraints apply can be determined through the API itself.

4. Implementation Approach

Beyond commercial add-ons such as *GrassHopper* and *Dynamo*, there exists numerous open-source APIs for visual programming such as *Node-RED, noflo, PyFlow* etc. Libraries such as *Node-Red* and *noflo* are based on JavaScript and can be deployed on the web, but they lack an interface for visualizing models, including IFC models. *PyFlow*, on the other hand, can be deployed as a stand-alone and also in conjunction with an open-source 3D modeller called *FreeCAD*. Thus, models can be visualised, information can be extracted from *FreeCAD* and used in visual programming. In this work, we have chosen to develop the SHACL module with *FreeCAD* and *PyFlow* so that information can be extracted from *FreeCAD* for easier modelling in *PyFlow*.

4.1 FreeCAD and PyFlow

FreeCAD is an open-source general-purpose modelling tool, with emphasis on parametric modelling(“FreeCAD: Your own 3D parametric modeller,” n.d.). Originally, it began with support for mechanical and product design, it now supports Architectural and Civil Engineering modelling also, by making use of *IfcOpenShell*. *FreeCAD* implementation closely resembles the work proposed by Preidel et.al, in which a Visual Code Compliance Language based on graphical notation is described (Preidel and Borrmann, 2016).

FreeCAD splits its tools into workbenches, thus making modularized parts, which can be mix-matched by users for creating models. All of these run on the scripting language Python, giving end-users flexibility to create their tools and functionalities. Additionally, it also supports 3rd party tools and libraries for a variety of applications, thus making it possible to run tools inside tools. One such tool is *PyFlow*, a general-purpose visual scripting framework for python (“wonderworks-software/PyFlow,” 2020). It contains editable node packages, which can be user-defined. A more customized version of the *PyFlow* is the *NodeEditor*, which customises the *PyFlow* nodes for interaction with files loaded in *FreeCAD*.

Figure 4.1 shows a general workflow of the SHACL Constraint creator using the above open-source applications. *FreeCAD*, the information visualization interface, interacts with *PyFlow* (the Visual programming interface). In the current set-up of *PyFlow*, two existing modules: an in-built *Pyflow module* and a *FreeCAD NodeEditor module* is pre-loaded. The in-built *PyFlow* module contains basic nodes for manipulation of any data. The *FreeCAD NodeEditor* contains pre-defined nodes for interacting with the *FreeCAD* environment. In this paper, a new module called “SHACL Constraint Creator” is added to the above modules in *PyFlow*. This module contains SHACL shape nodes which take minimal non-syntactical input from the user, to create

SHACL shapes(rulesets) based on information loaded in the *FreeCAD* module. As shown in the figure, all SHACL Shape nodes can take their base input (such as the Class to be checked, the property to be checked etc.) from the model loaded in *FreeCAD*.

In this paper, we take a use-case, where an IFC model “HelloWall.ifc” is loaded in *FreeCAD*. The wall has a property: *label* with value “Wall”. The other properties of this wall can be accessed in *PyFlow* giving this input to the node *FreeCAD_Object2 (refer to Figure 4.2)*. This node after execution is shown in Figure 4.3. This node is used as a reference for creating constraints using the *SHACL Constraints Creator module*, explained in the next section.

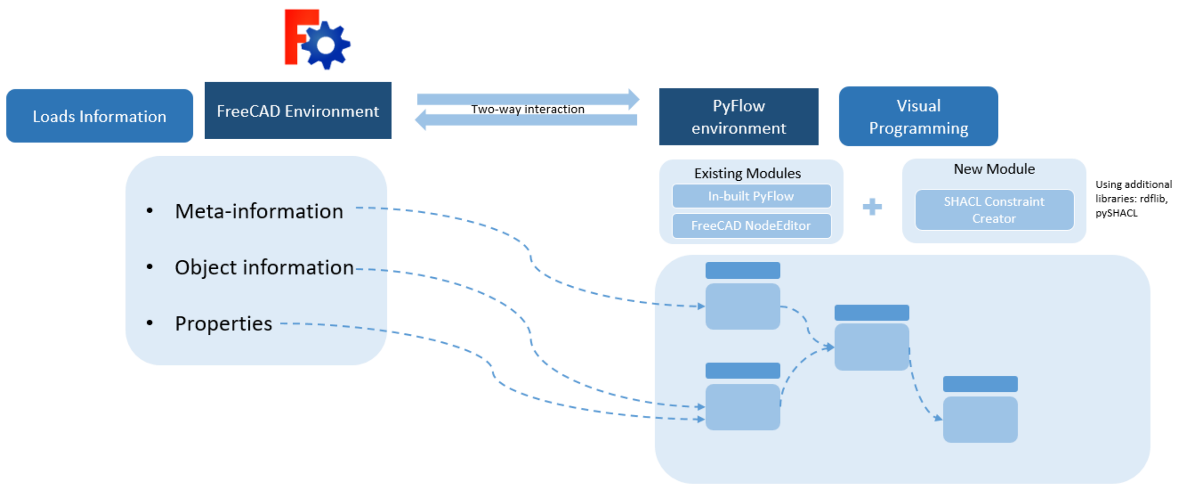


Figure 4.1: Implementation approach for SHACL Constraint Creator using FreeCAD and PyFlow

4.2 SHACL Constraints Creator module

A typical SHACL shape graph in *.ttl* serialization contains three parts: The first part defines the prefixes, followed by the testing node which defines the targeting class(Entity being checked) for the checking in the data graph, and finally the property for which the constraint is specified (property being checked). The *SHACL Constraints Creator module* is designed in the same structure.

The composition of a mvdXML ruleset, and a SHACL Shape is shown in Figure 4.4. An overview of how constraints in SHACL differs from constraints in mvdXML is shown in Listing 4.1. In this example, we check if in the entity *IfcWallCommon* there exists a value for the property *Thermal Transmittance*. First, the mvdXML-based formulation in *IFCDoc tool* is shown along with definition of the components of the file/s (Chipman et al., 2016). Below this, for the same example, the formulation in SHACL is shown.

```
Concept            <...> (excluded, refer to (Chipman et al., 2016)page 43 for
                          details)
Templates->


Concepts->         <Concept uuid="e9941408-82a6-4c00-a397-11087e6c5d1f" name="load

                      bearing external walls required to have property
                          'ThermalTransmittance'">

                       <Definitions>

                           <Definition>

                               <Body lang="de"><![CDATA[For all load bearing
                             external walls

                                the property 'ThermalTransmittance' shall be
                             applied]]></Body>

                           </Definition>

                       </Definitions>

                       <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>

                       <Requirements>

                           <Requirement applicability="export" exchangeRequirement=

                                  "ae70f764-938b-4cf7-9814-c29a47f56b0e"
                             requirement="mandatory"/>
```

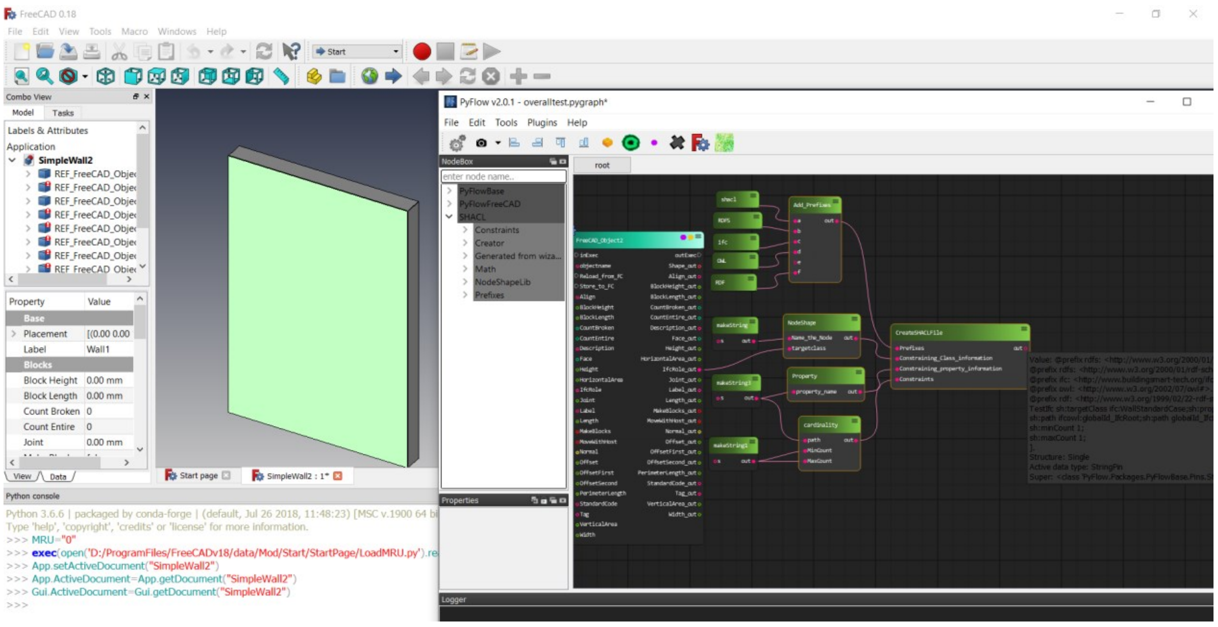
	`</Requirements>`
	`<TemplateRules operator="or">`
	`<TemplateRule`
Target Class->	`Parameters="O_PsetName[Value]='Pset_WallCommon' AND`
Target property->	`O_PName[Value]='ThermalTransmittance' AND`
	`O_PSingleValue[Exists]=TRUE"/>`
	`<TemplateRule`
Cardinality->	`Parameters="T_PsetName[Value]='Pset_WallCommon' AND`
	`T_PName[Value]='ThermalTransmittance' AND`
	`T_PSingleValue[Exists]=TRUE"/>`
	`</TemplateRules>`
	`</Concept>`
Required	`@prefix dash:<http://datashapes.org/dash#>.`
Prefixes->	`@prefix sh:<http://www.w3.org/ns/shacl#>.`
	`@prefix ifcowl:<http://www.buildingsmart-tech.org/ifcOWL/IFC2x3_TC1#>.`
	`@prefix inst:<http://www.linkedbd.net/resource11>.`
	`@prefix express:<http://www.w3id.org/express#>.`
	`@prefix rdf:<http:www.w3.org/1999/02/22-rdf-syntax-ns#>.`
	`@prefix xsd:<http://www.w3.org/2001/XMLSchema#`
	`@prefix owl:<http://wwww.w3.org/2002/07/owl#>.`
Definition of Sample shape->	`ifcowl:TestIfc`
	`a sh:NodeShape;`
Target Class->	`sh:TargetClass   ifcowl:IfcWallCommon;`
Target Property->	`sh:property [`
Cardinality->	`sh:path ifcowl:ThermalTransmittance;`
	`sh:minCount 1;`
	`sh:datatype xsd:integer`
	`].`

Listing 4.1: Comparison of rulesets in mvdXML, from IFCDoc tool(above)and SHACL shape file (below)

In the above example, *ifc:TestIfc* is a sample NodeShape in SHACL, which targets the entity *ifcowl:IfcWallCommon*, and specifies the property *ifcowl:ThermalTransmittance* is having a constraint that it should have at least one value and that value must be an integer (and not be empty). It has to be noted that the mvdXML file shown in Listing 4.1 is only a snippet, and the additional files which included the concept templates etc. will also have to be defined and created before checking.

Similar to the structure of SHACL shape in Listing 4.1, the SHACL Constraint Creator module in *PyFlow* contains a Prefix library, a *NodeShape library* and a *Constraints library*. While the *Prefix library* contains relevant prefixes, which the user can select, the *NodeShapeLib library* contains nodes which with input pins for the name of the sample node shape class, the targeted class for checking, and the property being checked.

In the *Constraints library*, value configurable nodes for checking cardinality, data type, relationship are defined. Figure 4.3 shows the *SHACL Constraint Creator module* in *PyFlow*. Upon running the workflow shown in Figure 4.3, a SHACL file is generated and saved, the contents of which are show in in Listing 4.2.



```
Python 3.6.6 | packaged by conda-forge | (default, Jul 26 2018, 11:48:23) [MSC v.1900 64 bi
Type 'help', 'copyright', 'credits' or 'license' for more information.
>>> MRU="0"
>>> exec(open('D:/ProgramFiles/FreeCADv18/data/Mod/Start/StartPage/LoadMRU.py').re
>>> App.setActiveDocument("SimpleWall2")
>>> App.ActiveDocument=App.getDocument("SimpleWall2")
>>> Gui.ActiveDocument=Gui.getDocument("SimpleWall2")
>>>
```

Figure 4.2: FreeCAD with sample file HelloWall.ifc loaded and PyFlow-the open source visual programming editor

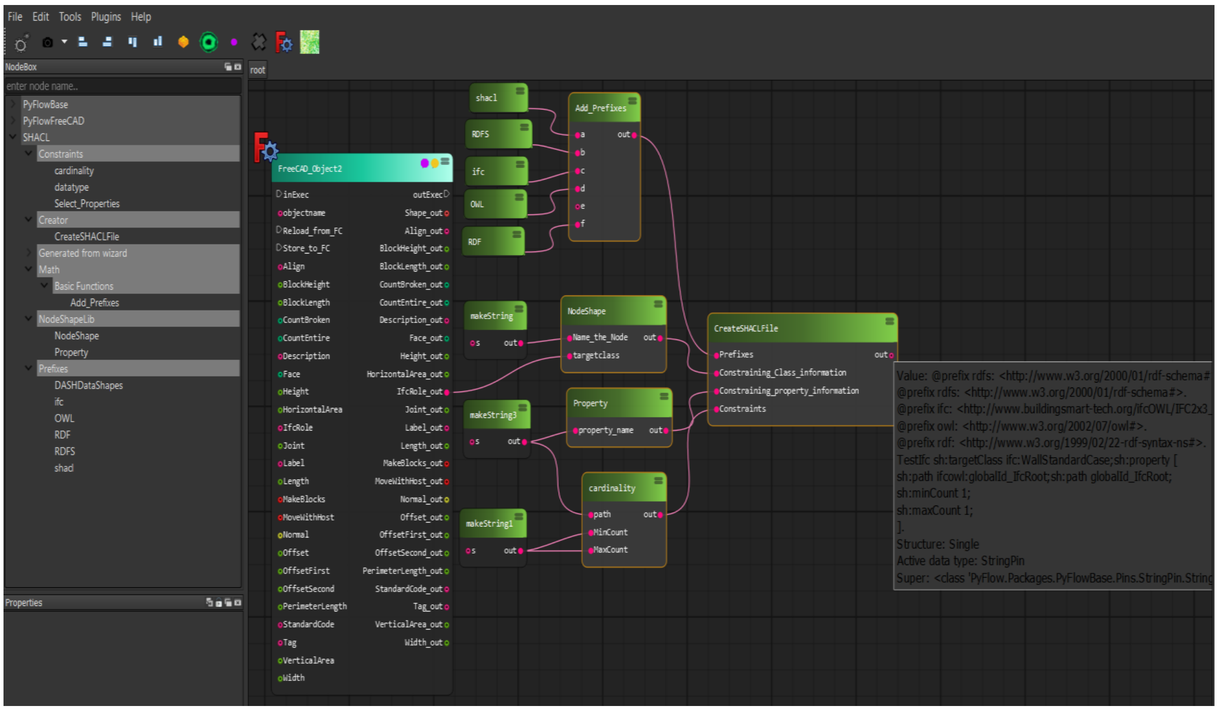
```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix ifcowl: <http://www.buildingsmart-tech.org/ifcOWL/IFC2x3_TC1#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

ifcowl:TestIfc sh:targetClass ifcowl:WallStandardCase;
sh:property [
sh:path ifcowl:globalId_IfcRoot;
sh:minCount 1;
sh:maxCount 1;
].
```

Listing 4.2: SHACL ShapeFile generated by the SHACL Constraints Creator

The nodes of *NodeShape* take input directly from the *FreeCAD_Object2* node (refer section 4.1). Additionally, an introspection feature is also implemented, which contains the IFC schema, which reads the relevant applicable inheritances and associated properties for the loaded object. Based on the information defined in the *FreeCAD_Object2* node, the search option displays only relevant *NodeShapes* and applicable property constraints, thus making it easier for the user to construct. All the nodes used in Figure 4.3 are reusable, meaning that they can be used as inputs for defining other constraints.

The SHACL Constraints Creator module is available on GitHub[6].



```
Value: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix ifc: <http://www.buildingsmart-tech.org/ifcOWL/IFC2x3_
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
TestIfc sh:targetClass ifc:WallStandardCase;sh:property [
sh:path ifcowl:globalId_IfcRoot;sh:path globalId_IfcRoot;
sh:minCount 1;
sh:maxCount 1;
].
Structure: Single
Active data type: StringPin
Super: <class 'PyFlow.Packages.PyFlowBase.Pins.StringPin.String
```

Figure 4.3: SHACL Constraint Creator module in PyFlow Environment

5. Discussion and Conclusion

Checking conformance of models finds application in situations beyond code compliance. As previously mentioned, such checking is necessary during file transfer between tools, formats and standards. The above situations are common-place when working in a collaborative environment, and hence access to an easy-to-use checking tool is necessary for all stakeholders involved.

From a stakeholder perspective, who has to create these constraints, the main task is to convert the text-based requirements into computer-executable rule-sets. The *SHACL Constraint Creator* facilitates this through an interface in which the user can drag and drop nodes, and populate the nodes with information from the text-based requirements. The computer-executable rulesets are then automatically generated from the information, which can then be used to validate any file. The *FreeCAD* environment supports multiple formats, including images, IFC, CAD/DWG etc., and hence constraints can be created for any type of information, thus it can be used for also checking the information in a linked data environment. In the example discussed in section 4.2, the model loaded in *FreeCAD* is a *.ifc* file, while the SHACL constraints are created for *ifcOWL*, to demonstrate the viability of using SHACL for linked building data checking. It has to be noted that the current implementation is still under development.

SHACL also supports querying, with SHACL Advanced Features focusing on SPARQL queries for extensions. Additionally, efforts are on for incorporating GraphQL with

[6] https://github.com/sbalot/SHACLConstraintCreator

SHACL("Publishing RDF/SHACL Graphs as GraphQL," n.d.; Taelman et al., 2019). Such querying can also be modularised as nodes (similar to the way constraints are in this paper) so that end-user directly inputs informal text-information, which is then converted to queries or used for further validation. FreeCAD also contains modules for connecting to BIMServer[7] and a BCF tool[8] for uploading/downloading and updating files in it. Further, if *PyFlow* can be wrapped with JavaScript or used with container solutions such as Docker, it can be deployed as a standalone on the web, thus enabling easier creation of constraints. Future work will be focusing on incorporating the functionalities of these in *SHACL Constraints Creator*.

Acknowledgement

This research was funded through a doctoral research grant by the *Deutscher Akademischer Austauschdienst (DAAD)*.

References

Beetz, J., Leeuwen, J. van, Vries, B. de, (2009). IfcOWL: A case of transforming EXPRESS schemas into ontologies. AI EDAM 23, pp.89–101. https://doi.org/10.1017/S0890060409000122

Catarci, T., Santucci, G., (1995). Are Visual Query Languages Easier to Use than Traditional Ones? An Experimental Proof. BCS HCI, pp. 323–338.

Chipman, T., Liebich, T., Weise, M., (2016). mvdxml: Specification of a standardized format to define and exchange Model View Definitions with Exchange Requirements and Validation Rules. Model Support Group (MSG) of buildingSMART International Ltd.

FreeCAD: Your own 3D parametric modeler [WWW Document], n.d. URL https://www.freecadweb.org/ (accessed 2.28.20).

Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, J., De Meyer, R., Van de Walle, R., Van Campenhout, J., (2011). A semantic rule checking environment for building performance checking. Automation in Construction 20, pp.506–518. https://doi.org/10.1016/j.autcon.2010.11.017

Preidel, C., Borrmann, A., (2016). Towards code compliance checking on the basis of a visual programming language. Journal of Information Technology in Construction (ITcon) 21, pp.402–421.

Publishing RDF/SHACL Graphs as GraphQL [WWW Document], n.d. URL https://www.topquadrant.com/graphql/shacl-graphql.html (accessed 3.8.20).

Roxin, A., (2016). A Semantic Web Approach for defining Building Views.

Taelman, R., Sande, M.V., Verborgh, R., (2019). Bridges between GraphQL and RDF, in: W3C Workshop on Web Standardization for Graph Data. Presented at the W3C Workshop on Web Standardization for Graph Data, p. 4.

van Strien, E., (2015). MVD Checker Guide.pdf. Technische Universiteit Eindhoven, Eindhoven, The Netherlands.

Weerink, J., (2016). Verifying the completeness of Building Information Models. Technische Universiteit Eindhoven, Eindhoven, The Netherlands.

wonderworks-software/PyFlow [WWW Document], 2020. URL https://github.com/wonderworks-software/PyFlow (accessed 2.28.20).

Zhang, C., Beetz, J., Weise, M., (2015). Interoperable validation for IFC building models using open standards 20, pp.24–39.

[7] https://www.freecadweb.org/wiki/Arch_BimServer
[8] https://github.com/podestplatz/BCF-Plugin-FreeCAD