Model Checking on the Semantic Web: IFC Validation Using Modularized and Distributed Constraints

Chi Zhang, c.zhang@tue.nl
Eindhoven University of Technology, The Netherlands

Jakob Beetz, j.beetz@tue.nl
Eindhoven University of Technology, The Netherlands

Abstract
In this paper we present a method for IFC instance validation based on semantic web technologies. This method is designed to facilitate reusing rules in order to simplify rule structuring processes. It aims to improve the modularity and reusability of validation constraints with an open and expressive method. A prototype model checking environment is implemented based on open source APIs and reasoning engines. Based on real BIM use-cases, requirement ontologies and related inference rules are defined to derive information from IFC instances to make them suitable for model checking. Checking constraints in these use-cases are classified according to their underlying logics to build a set of templates. All the ontologies, templates, inference rules and checking constraints are maintained as RDF graphs and can be published and distributed as web resources. Based on these experiences, a detailed discussion about added values, limitations and application strategies is also provided.

Keywords: Industry Foundation Classes, Ontology, Inference Rules, Constraints, Templates， SPARQL

1 Introduction

The building industry requires qualified information to facilitate collaboration and automation between multiple domains and parties involved in construction projects. The Industry Foundation Classes (IFC) are considered the most suitable standard to approach this objective (Eastman et al 2011). As a general data model, however, IFC provides the flexibility required across different domains and processes but results in lacking constraints for specific tasks. The quality of building models highly relies on knowledge and personal experiences of domain experts and software implementers, which often results in models insufficiently rigid for computational environments.

In order to address this issue, additional rules need to be defined to check and validate IFC models. These rules can be, from general to specific, constraints in model view definitions across multiple domains e.g. the Coordination View, to company- and project-specific business rules. They include constraints on explicit data captured in IFC instances e.g. required entities and properties, as well as higher level constraints and regulations for modeling and buildings, which usually need reasoning implicit information from static data. Interpreting these constraints initially provided in natural languages and structuring them into computer-processable rules is a time-consuming process that requires a profound technical understanding of the IFC model specification and programing work. Therefore, developed rules should be easily reused to simplify this process, which introduces additional demands for rule definition methods. These requirements for rule languages can be categorized into two sub-issues: 1) in a functional aspect, the methods should contain technical layers or special mechanisms to formally define and maintain sharable concepts, constructs and rules; 2) concerning applications, they need to be open and platform-independent, since they are much easier for reusing, public assessment and parallel working (Eastman et al 2009).

In the research reported upon in this paper, we propose to use semantic web technologies to achieve aforementioned objectives. In the following section, the background and a brief review of related researches are provided. In section 3, the overall framework of this semantic web based approach is presented. With use-case examples and implementations reported in section 4 and 5, a detailed discussion about added value, limitations and future work concludes this paper.

2 Background and Related Work

In order to improve the interoperability for IFC-based working processes, the methodology of Model View Definition (MVD) addresses the issues described in section 1 from the perspective of software implementations (IUG, 2012). As a subset of the IFC data model, an MVD specifies the needed information and its representations as agreements for multiple software vendors. The export-import interfaces are developed between IFC models and BIM authoring tools according to specific MVDs. To validate implementations, interfaces are tested against MVDs as a part of BIM certification procedures (Hausknecht et al 2014). This approach evaluates and improves the quality of IFC implementations, but the *contents* of models created by end-users still needs to be checked.

A number of BIM standards and company- or project-specific requirements have been developed in parallel to the MVD approach. These documents are sets of business rules to specify more detailed requirements for domain end-users. Some of these requirements are more related to data representations, which are very specific in their demands regarding the existence or value limitations directly on IFC entities and properties. Others contain regulations according to domain knowledge. Both of these need to check IFC files against specific requirements and rules. Compared to BIM certification services, which are often provided by international organizations and usually focus on validations according to stable MVDs, regional, company- and project-specific BIM standards and agreements are heterogeneous and diverse and require a higher level of customization.

The common approach for automated model checking is to systematically compare the information in models and the constraints in requirements (Dimyadi and Amor 2013). Since IFC models are assumed as the input of building models, the challenges are a) how to properly derive potentially non-explicit information from models and b) how to digitally represent checking requirements. In the tools' realm, many researchers have suggested that the digital representations of requirements should be separated from processing programs (Eastman et al 2009, Hjelseth 2010). There are very few open technologies to support this process. The EXPRESS and EXPRESS-X are ISO standards, which can be used to define constraints for IFC. They have been implemented in Jotne EDM Model Server and IfcCheckingTool from KIT, the latter of which is used for BIM certification applied by buildingSMART (Hausknecht et al 2014). The evolving mvdXML specification is currently the only open standard used to formally capture MVDs and validating IFC instances (Chipman et al, 2013). Its built-in constructs `ConceptTemplate` and `Concept` are used to develop reusable subsets and constraints. However, these modelling constructs specify constraints on low-level IFC entities and attributes in its underlying schema, which make them more suitable for BIM certification and are not easily reusable by the general public.

Many concepts used in the checking requirements such as domain-specific relationships and property sets are specified outside the scope of the IFC schema. Therefore, to make models suitable for checking and facilitate structuring constraints, one of the critical questions is how to specify and enhance the semantics for IFC models. Recent trends using ontology and semantic web approaches to consistently embed domain knowledge into rule definitions is a step in the right direction (Solihin and Eastman 2015). An ontological method is proposed by Venugopal et al 2012 to provide a high level specification for IFC schema in the precast concrete domain. By converting IFC to OWL, external semantics are embedded into IFC models by Beetz et al 2009. Based on this approach, a prototype checking environment is presented by Pauwels et al 2011 with an example implementation about acoustic performance checking. Some domains outside of the building industry have also focused on this direction and have more standardized solutions. The ISO 15926-2 data model (also EXPRESS based), which is the IFC equivalent in the gas and oil industry, uses the predicate logic to formalize its semantics by mapping the elements to external concepts and templates in ISO15926-7. It is partly implemented by OWL constructs in ISO 15926-8.

Many of the existing efforts for semantic specification and enhancement in the building industry did not consider on model checking. The checking environment proposed by (Pauwels et al

2011) presented the expressivity of semantic web technologies but did not consider on maintenance and reusability of constraints and is based on an initial set of limited use-cases. The ISO 15926 provided an example for officially formalizing semantics and constraints for EXPRESS based models, which is very important for their reusability. Some of its basic ideas and technologies used can be reused by the building industry.

3 Methodology

The model checking process provides rules and "asks" questions to knowledge systems to get pass or fail answers. Many of these rules are shared by different rule-sets. Each of the rules can be interpreted by predicate logic with specific concepts and underlying logics, which are often referenced and shared by other rules. For example, the requirements "a building should contain at least one storey" and "a storey should have at least two exits" share the concept "storey" and the same logic. This example shows that not only the rules themselves should be reusable, but that all concepts and logics should be reusable across different rules. The concepts are used to encapsulate derived information from IFC instances, and the reusable logics are to simplify structuring rules. The presented method defines concepts and logics into separate structures. The framework has five conceptual layers (Fig. 1). In the following part of this paper, "rules" only represents inference rules that are used to derive new facts before the checking execution, while setting restrictions for the checking process is described as "constraints".

- Layer 1: the IfcOWL ontology as the baseline. To utilize the semantic web technologies, the EXPRESS schema of IFC is converted to OWL, and the IFC instances are converted to RDF triples (Beetz et al 2009, Törmä 2014). The IfcOWL ontology supports inferences based on a subset of the semantics in IFC schema to generate new facts from IfcRDF instances. For example, an `IfcWallStandardCase` will also be interpreted as an `IfcWall`.
- Layer 2: Ontologies which define concepts used in requirements. The terminologies used in checking rules are categorized and structured as ontologies, including classes, relationships and properties. These ontologies are developed in OWL and can be distributed and interlinked as web resources with different URIs and namespaces according to the authors' policies and requirements.
- Layer 3: Inference rules that reference concepts in Layer 1 and 2. They are used to convert data captured by IfcRDF instances to information represented by concepts in Layer 2. This layer ensures that Layer 1 and 2 can be developed independently from each other, and any updates in one of them only affects rules in Layer 3.
- Layer 4: Constraint templates. All the requirements have two parts: 1) the context the constraints should be applied to, and 2) the properties to which the constraints should be applied (Eastman et al 2009). These if-then logics are encapsulated as reusable templates with arguments in this layer.
- Layer 5: Constraint instances. This layer contains constraint instances which are derived from respective natural-language documents. Each of them has an "is-a" relationship with a template in Layer 4, and the arguments of the template are replaced either by specific concepts in Layer 2 and 1 or by simple data values and are then maintained as constraint instances. This layer ensures that constraints can be reused across different rule sets.

Each of the five layers are represented in RDF, and all the concepts, rules, templates and constraints are maintained as web resources with their own URIs so they can be referenced, interlinked and reused. Layer 1, 2 and 4 are developed independently from each other. With inference rules and reasoning engines, data within IfcRDF instances is partially converted to information represented by concepts in Layer 2 to be more suitable for model checking. End-users can (re-)use concepts in Layer 2 and 1 and constraint templates in Layer 3 to develop specific constraints, which are also shared by different rule-sets.

Besides these five layers, existing vocabularies from RDF, OWL, SPARQL and SPARQL Inferencing Notation (SPIN) are used. SPIN is currently a W3C submission used to represent SPARQL rules on the semantic web (SPIN, 2011). It captures SPARQL queries as RDF representations, and also enhances the expressivity by a meta-modelling vocabulary which supports defining additional operators, functions and query templates. In this framework, Layers 3, 4 and 5 are developed using this technology. The requirements in Statsbygg BIM manual (Statsbygg 2011) are used as use-cases and validation in the research presented here.

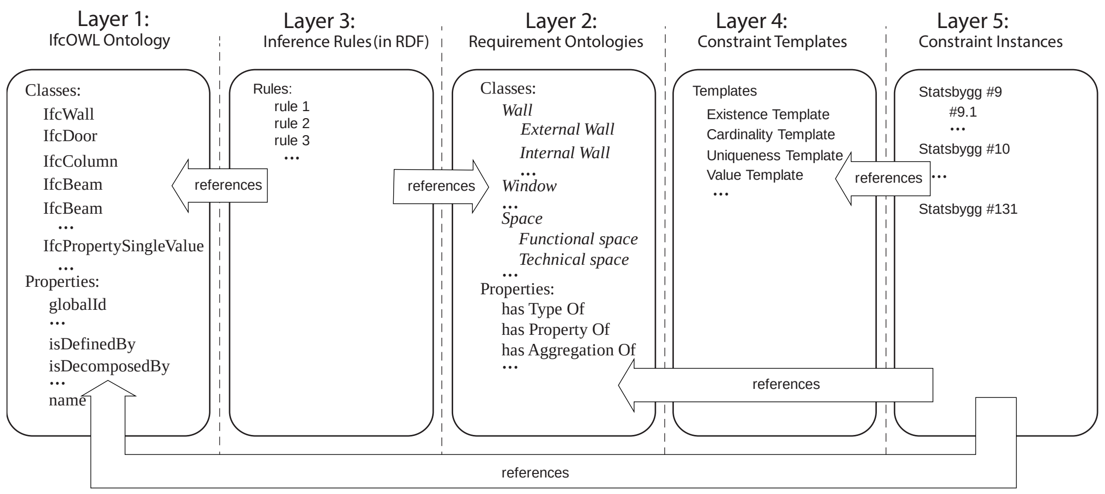

Figure 1 Conceptual architecture of proposed approach

3.1 Layer 2: Requirement ontologies

The Statsbygg BIM manual consists of 131 requirements in total. In our experiments, we manually interpret the requirements to semi-structured sentences to markup the "if-then" structures, classes, properties, value limitations and operators. The classes and properties are categorized to structure an ontology. We manually interpreted their semantics from natural language to decide on the types for the concepts. For example, most relationships such as "has_Aggregation_Of" are defined as object properties instead of objectified relationships as they are defined in the IFC schema e.g. `IfcRelAggregation`. Statsbygg is a standard that already specifies many requirements using explicit IFC entities and attributes, but still many additional concepts e.g. "external wall", "fire compartment" are used that have no immediate equivalent in the IFC model. To explicitly formalize these additional concepts is the main purpose of the ontology on Layer 2. To handle of property sets used by many requirements across different use-cases we have opted for the following modeling constructs: to define them as classes that have local names starting with upper case in our ontology, and each of them also has a lower case version as a data type property. An example of the `IsExternal` property is presented in equations (2) and (3).

3.2 Layer 3: Inference rules

The inference rules are built up between requirement ontologies and IfcOWL ontology. These rules are used to convert IFC data to models that are suitable for checking. The purpose is specified as follows. In the logic formulas of this subsection, the concepts in Layer 1 are set in **bold** to differentiate them with concepts in Layer 2, which are set in *italic*.

- Shortcut for commonly used IFC structures. In the IFC 4 documentation, commonly used IFC structures have been listed as "fundamental concepts", which are recommended for use in IFC models for general scenarios. Applying inference rules, these structures can be encapsulated into simplified structures to make them closer to natural language descriptions and more convenient for querying data. A typical example is the `IfcProperty` construct, which is simplified to a triple relationship from the object to an `IfcProperty`, which is further converted to a data type property in equations (1), (2) and (3). Other common structures which are related to our use cases are also encapsulated.
- Classifying IFC elements according to domain concepts and knowledge. Many checking requirements regulate constraints with domain concepts e.g. external wall, entry point, which are not typed in the IFC schema. Inference rules identify these concepts in IFC instances to make them suitable for model checking. An example is presented in equations (4) and (5).
- Reasoning new facts from data captured in IFC instances to enhance objects with additional properties and relationships. The derivation of new facts is based on the functions within

inference rules. Using e.g. arithmetic operations on numeral types or logic reasoning, additional properties and relationships are inferenced to enrich objects.

$$\begin{gathered}\mathbf{IfcObject}(x)\wedge\mathbf{isDefinedByProperties}(x,z)\wedge\mathbf{IfcRelDefinesByProperties}(z)\\ \wedge\mathbf{RelatingPropertyDefinition}(z,w)\wedge\mathbf{IfcPropertySet}(w)\wedge\mathbf{hasProperties}(w,y)\\ \wedge\mathbf{IfcProperty}(y)\rightarrow hasPropertyOf(x,y)\end{gathered} \tag{1}$$

$$\mathbf{IfcProperty}(x)\wedge\mathbf{name}(x,\text{"IsExternal"})\rightarrow IsExternal(x) \tag{2}$$

$$hasPropertyOf(x,y)\wedge IsExternal(y)\wedge\mathbf{nominalValue}(y,z)\rightarrow isExternal(x,z) \tag{3}$$

$$\mathbf{IfcElement}(x)\wedge isExternal(x,true)\rightarrow External\ Element(x) \tag{4}$$

$$External\ Element(x)\wedge\mathbf{IfcDoor}(x)\rightarrow External\ Door(x) \tag{5}$$

These rules are defined in SPARQL `CONSTRUCT` statements, converted into SPIN and associated with specific types through `spin:rule` construct.

3.3 Layer 4: Constraint templates

The constraint templates are used to encapsulate checking logics for rules. Each template has arguments that can be substituted by concepts in Layer 2 and locally defined values to create constraint instances. Many of the data is converted to triples in RDF and the checking process becomes more straightforward. For each type of templates, a table is provided with description, examples, logic semantics and SPIN templates in the SPARQL syntax. These templates are developed with `ASK` and `CONSTRUCT` statements, and maintained as instances of `spin:Template`.

3.3.1 Existence and cardinality constraints

The constraints of this type are used to require the existence or quantities of properties or relationships for specific classes. Some of the examples are specified in Table 1.

Table 1 **Template table for existence and cardinality**

	Existence	Type Existence	Cardinality
Description	Check the existence of an attribute/property for a class	Check the existence of an attribute/property which belongs to a specific type for a class	Check the quantities of an attribute/property which belongs to specific type for a class.
Example	"If it is a stair, then a name attribute should be provided"	"If it is a building, then it should have a property "GrossVolume"	"If it is an indoor space, then it should be included in 1 and only 1 fire compartment"
Logic semantics	$C(s)\supset\exists o\big(p(s,o)\big)$	$C(s)\supset\exists o\big(p(s,o)\wedge D(o)\big)$	$C(s)\supset\exists_{\geq m}^{\leq n}o\big(p(s,o)\wedge D(o)\big)$

SPIN template (SPARQL syntax)	**CONSTRUCT** {
	  _:b0 a spin:ConstraintViolation .
	}
	WHERE {
	  ?s a **?C** .
	  FILTER NOT EXISTS {
	    ?s **?p** ?o .
	  } .
	}
		CONSTRUCT {
		  _:b0 a spin:ConstraintViolation .
		}
		WHERE {
		  ?s a **?C** .
		  FILTER NOT EXISTS {
		    ?s **?p** ?o .
		    ?o a **?D** .
		  } .
		}
			CONSTRUCT {
			  _:b0 a spin:ConstraintViolation .
			}
			WHERE {
			  ?s a **?C** .
			  FILTER (
			  (op:count(?s, **?p**, **?D**) > **?n**) ||
			(op:count(?s, **?p**, **?D**) < **?m**)) .
			}

Arguments	C, p	C, p, D	C, p, D, m, n

3.3.2 Value constraints

This type of constraint is defined to restrict the value of a property. They usually restrict the value range for numeric data, or format naming conventions for Strings. The "operator" in Table 2 is a function such as “equals” or “greater_than” for numeric data, “starts_with”, “contains” or “regex” for strings etc. Some SPARQL implementations allow the extension of such operators with e.g. domain-specific spatial reasoning functionality.

Table 2 **Template table for value**

	Value
Description	Check the value of a simple data type attribute/property for a class
Example	“If it is a site, then its ‘land title number’ shall contain a cadastral number”
Logic semantics	$C(s) \supset \exists o\big(p(s,o) \wedge operator(o,a)\big)$

SPIN template (SPARQL syntax)	**CONSTRUCT** {
	  _:b0 a spin:ConstraintViolation .
	}
	WHERE {
	  ?s a **?C** .
	  FILTER NOT EXISTS {
	    ?s **?p** ?o .
	    FILTER operator(?o, **?a**) .
	  } .
	}

Arguments	C, p, operator, a

3.3.3 Uniqueness

This type of constraints define global uniqueness for simple data type properties or local uniqueness for aggregation type properties. The global uniqueness is specified in Table 3 with an example.

Table 3 **Template table for global uniqueness**

	Global Uniqueness
Description	Check the global uniqueness of a simple data type attribute/property for a class in the model.
Example	“If it is a type object, then it shall have a unique name”
Logic semantics	$C(s) \supset \neg\exists a(p(s,o) \wedge C(a) \wedge p(a,b) \wedge equals(o,b) \wedge \neg equals(s,a))$

SPIN template (SPARQL syntax)	**CONSTRUCT** {
	  _:b0 a spin:ConstraintViolation .
	}
	WHERE {
	  ?s a **?C** .
	  ?a a **?C** .
	  ?s **?p** ?o .
	  ?a **?p** ?b .
	  FILTER ((?o = ?b) && (?s != ?a)) .
	  }

Arguments	C, p

3.3.4 Complex constraints

All of the aforementioned templates are constraints for single subject, predicate and object triple. However, there are requirements that restrict data on more complex graph structures and have more sophisticated logics (e.g. if-then conditional, exceptions) which cannot be covered by them. There are two strategies to make templates for such requirements.

The first strategy is to make plain templates which encapsulate complex logics. Experiments with the use-case showed that most complex templates are still reusable by many constraints using this first strategy. For example, the checking logic “if some slabs are in the same type, then their thicknesses and materials should be the same” is also reusable by checking the consistency for other types of building elements. However, there are some examples like “if it is a ‘functional space’, then its ‘height’ should be ‘smaller than’ the ‘distance’ between the ‘upper edge’ of the ‘floor slab’ of the ‘storey’ and ‘lower edge’ of the ‘floor slab’ of the ‘storey’ ‘above’,” which have 9 distinct concepts.

This checking logic still can be defined as a plain template, which however is hardly reusable by other constraints.
The second strategy is to use nested templates for this type of constraints, within which each SPARQL subquery is defined by a simple template, which can be reused by other complex templates. The second strategy needs further testing and require extensions on the current SPIN implementation.

4 Implementation of a checking environment

The on-going implementation is based on the open source Apache Jena and SPIN libraries and APIs (SPIN, 2011). The architecture and data flow of the checking process are illustrated in Fig. 2. Layers are developed with different namespaces.

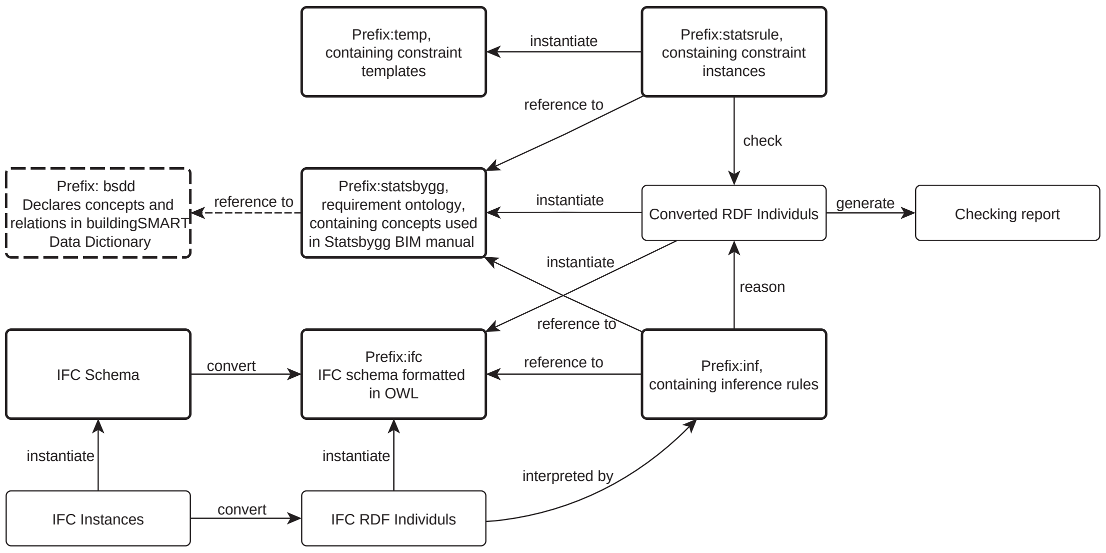
Prefix:temp, containing constraint templates
instantiate
Prefix:statsrule, constaining constraint instances
reference to
check
Prefix: bsdd Declares concepts and relations in buildingSMART Data Dictionary
reference to
Prefix:statsbygg, requirement ontology, containing concepts used in Statsbygg BIM manual
instantiate
Converted RDF Individuls
generate
Checking report
instantiate
reason
reference to
IFC Schema
convert
Prefix:ifc IFC schema formatted in OWL
reference to
Prefix:inf, containing inference rules
instantiate
instantiate
interpreted by
IFC Instances
convert
IFC RDF Individuls

Figure 2 Implementation of a prototype model checking environment

In this checking environment, end-users choose constraint templates in `temp:` and arguments in `statsbygg:` and `ifc:` to define constraints. Each of the templates has a `spin:labelTemplate` property, which explains the checking logics in natural language. For example, the Type Existence template specified in 3.3.1 has this property of "A {?C} should {?p} {?D}", which is used for the user interface and checking report format. These three arguments can be respectively assigned to compose a constraint. For example, the requirement "Statsbygg 48" introduced in section 3.3.1 is maintained in RDF (Turtle syntax) as follows.

```
statsrule:48  a            temp:TypeExistence ;
                    arg:C      ifc:IfcBuilding ;
                    arg:p       statsbygg:has_Property_Of ;
                    arg:D      statsbygg:GrossVolume  .
```

5 Discussion

5.1 Added value of presented research

In this method proposed here, all the ontologies, inference rules, constraint templates and instances are maintained as distributed web services, represented in RDF/XML or other RDF notations and published as linked data (Heath and Bizer 2011). It provides a web-based environment to make sharable rules for model checking. It improves the granularity from the constraint level to the level of concepts and logics within constraints. All these resources can be reused by referencing their URIs. All the rules and constraints are completely declarative, so it is relatively easy to create a user interface close to natural language representations. Domain end-users can use the concepts and constraint templates to define their own rules, but defining inference rules and new templates still need technical knowledge about SPARQL and the IFC schema. As the OWL version for IFC 4 has been proposed as a standard, this approach could be a use-case for this ontology (replace the current Layer 1) and be more standardized in the future (Pauwels 2014).

The existing SPARQL vocabularies and built-in functions make this method highly expressive. Additional domain specific functions can be developed through `spin:Function` with SPARQL and JavaScript. In our current experiments, around 80 percent of the requirements in Statsbygg BIM manual are covered.

The modularity of the proposed method reduces the redundancy in the system and simplifies the verification work for checking results. The inference rules and constraints can be tested independently. In the constraint aspect, once a constraint template is verified, it will not be modified and its instances do not need verification again.

To standardize the terminologies and relationships in the Statsbygg requirements ontology, a potential extension of current framework is to use a standard library such as bSDD as the external references for these concepts (Fig. 2). For example, the `statsbygg:External_Wall` has an `owl:equvalentClass` relationship with the buildingSMART Data Dictionary concept `bsdd:3vHZS2oT0Hsm00051Mm008` (BuildingSMART 2014). Other requirements ontologies can reference to the same standard to align them in order to make more consistent representations for concepts.

5.2 Limitations and lessons learned

The current implementation aims to set up a platform, which still needs extensions of additional inference rules, templates and functions. For example, as described in section 3, an external wall is mapped to the explicit “IsExternal” property in the respective PSet, but is not based on reasoning of topological relationship between the wall and the building. Similar examples like the relationship “has_upper_floor” are based on relatively explicit information defined in IFC instances, but not directly based on geometry and location data. This part needs further investigation and will be enhanced in the inference rules in Layer 3.

The implementation is based on the reasoning engines of Jena and Topbraid SPIN. The SPIN implementation uses a forward chaining approach, which draws all inferences it can from rules. However, it does not automatically isolate rules which are not needed for the constraint checking. Considering performance, backward chaining approaches are needed. This results in a dilemma that either using another technology which brings better performance in the inference part or limiting it to a single platform.

Another issue arises for complex constraints, which are mentioned in 3.3.4. Only very few of our use cases need complex constraints. Plain templates may have issues concerning reusability for such constraint templates. A potential solution is to use nested templates, which need extensions for the SPIN platform. More use-cases are needed for evaluation and testing.

6 Summary and Future Work

In this paper, we present the use of semantic web technologies to check and validate IFC models. It aims to improve the modularity and reusability of validation constraints with an open, sharable and expressive approach. Three directions to extend the current work have been identified: We are going to expand the scope of use-cases and apply more domain-specific requirements to further validate this method. More domain functions, templates and inference rules will be added. Another direction is to simplify the development for inferencing rules by providing templates for inference rules using the same method in Layer 4. A classification for inference rules should be developed. The third direction is to further investigate the technical potential for addressing the issues regarding performance and complex constraints specified in subsection 5.2. Validation of this method will be conducted by domain end-users to reuse developed concepts and templates.

References

Beetz, J., van Leeuwen, J.P. and de Vries, B. (2009). "IfcOWL: A case of transforming EXPRESS schemas into ontologies." Artificial Intelligence for Engineering Design, Analysis and Manufacturing. vol. 23. no.

Special Issue 01. 89-101.

BuildingSMART. (2014). buildingSMART Data Dictionary. http://bsdd.buildingsmart.org/.

Chipman T., Liebich T. and Weise M. (2013). 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..

Dimyadi J., Amor R. (2013). Automated building code compliance checking –where is it at?. *Proceedings of the 19th CIB World Building Congress*, Brisbane 2013: Construction and Society. 172-185. 2013.

Eastman C., Lee J., Jeong Y. and Lee J. (2009). Automatic rule-based checking of building designs, *Automation in Construction*, 18 (2009) 1011-1033.

Eastman C., Teichol P., Sacks R. and Liston, K. (2011). *BIM handbook –a guide to building information modeling for owners, managers, designers, engineers, and contractors*, 2nd edition, John Wiley & Sons Inc.

Hausknecht K., Liebich T., Weise M., Linhard K., Steinmann R., Geiger A., Hafele K.-H. (2014). BIM/IFC software certification process by buildingSMART, *eWork and eBusiness in Architecture, Engineering and Construction*, CRC Press, 129-133.

Heath T. and Bizer C. (2011) *Linked Data: Evolving the Web into a Global Data Space* (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, 1-136. Morgan & Claypool.

Hjelseth E., Nisbet N. (2010). Overview of concepts for model checking. *CIB-W078 Conference in Cairo*, pp. 16-18. 2010.

IUG, International User Group of buildingSMART International Ltd. (2012). An integrated process for delivering IFC based data exchange. http://iug.buildingsmart.org/idms/methods-and-guides/Integrated_IDM-MVD_ProcessFormats_14.pdf/view, accessed December 2012.

Jeong Y.-S., Eastman C. M., Sacks R. and Kaner I. (2009). Benchmark tests for BIM data exchanges of precast concrete, *Automation in Construction*, 18, pp469-484.

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(5), 506–518.

Pauwels, P. (2014). ifcOWL ontology file added for IFC4_ADD1. Available on https://www.w3.org/community/lbd/2014/12/12/ifcowl-ontology-file-added-for-ifc4_add1/.

Solihin, W., & Eastman, C. (2015). Classification of rules for automated BIM rule checking development. *Automation in Construction*, 53, 69-82.

Statsbygg, (2011). Statsbygg Building Information Modelling Manual Version1.2. Available at: http://www.statsbygg.no/bim, accessed January 2014.

SPIN. (2011). SPARQL Inferencing Notation. http://spinrdf.org/.

Torma, S. (2014). Opening BIM to the Web – IFC-to-RDF Conversion Software. Available at: http://rym.fi/results/opening-bim-to-the-web-ifc-to-rdf-conversion-software/.

Venugopal, M., (2011). Formal Specification of Industry Foundation Class Concepts using Engineering Ontologies. Ph.D. thesis, Georgia Institute of Technology, Atlanta.