

BimSPARQL: Domain-specific functional SPARQL extensions for querying RDF building data

Chi Zhang^{a,*}, Jakob Beetz^b

^a *Department of Built Environment, Eindhoven University of Technology, P.O. box 513, Eindhoven, The Netherlands*

E-mail: c.zhang@tue.nl

^b *Department of Architecture, RWTH University Aachen, Schinkelstrasse 1, 52062 Aachen, Germany*

E-mail: j.beetz@caad.arch.rwth-aachen.de

Abstract. In this paper, we propose to extend SPARQL functions for querying Industry Foundation Classes (IFC) building data. The official IFC documentation and BIM requirement checking use cases are used to drive the development of the proposed functionality. By extending these functions, we aim to 1) simplify writing queries and 2) enhance query abilities to retrieve useful information implied in 3D geometry data according to requirement checking use cases. Extended functions are modelled as RDF vocabularies and classified into groups for further extensions. We combine declarative rules with procedural programming to implement extended functions. Real use cases and building models are used to demonstrate the value of this approach and indicate query performance. Compared with query techniques developed in the conventional Building Information Modeling domain, we show the added value of such approach by providing an extended application example of querying building and regulatory data, where spatial and logic reasoning can be applied and data from multiple sources are required. Based on the development, we discuss the applicability of proposed approach, current issues and future challenges.

Keywords: BimSPARQL, IFC, ifcOWL, SPARQL, function

1. Introduction

As integrating data in the architecture, engineering and construction (AEC) industry is becoming increasingly important [47], Building Information Modeling (BIM) has been adopted by a growing number of industry practitioners and has led to the specification and standardization of the data standard Industry Foundation Classes (IFC) [15,26]. Using BIM applications and the IFC standard to create, exchange and process building-related data is the state-of-the-art in the AEC industries' day-to-day operations. As many researchers have discussed however, the use of the data modeling and exchange technologies underlying the

IFC format, the ISO 10303 family of standards, referred to as STEP (STandard for the Exchange of Product model data) [42] has a number of limitations that are partially rooted in the Closed World Assumption (OWA) nature and the limited semantics of the underlying meta-models [4,14,36]. Even using IFC-based instance building models, the retrieval of domain specific information is currently challenging for industry practitioners who are generally depending on proprietary, vendor-specific solutions. Building models are used for different engineering tasks, where information needs to be derived according to a wide range of use case requirements. However, the IFC data model is designed for the creation and exchange of product data, but not tailored for various query and analysis tasks. Many useful relationships and properties that are ex-

*Corresponding author. E-mail: c.zhang@tue.nl.

plicitly defined or implied in building models are difficult to retrieve in day-to-day processes. Furthermore, the IFC meta-model is limited by its schema which is not flexible enough to adapt to situations when data from different sources needs to be integrated and processed. Although IFC is a data model aiming to cover the entire AEC industry, much information used in common industry scenarios is not specified within the scope of the IFC meta model, including e.g. product classifications, building requirements and regulations as well as data from neighboring industries such as urban planning and sensor networks.

Using the Resource Description Framework (RDF) and Semantic Web technologies to represent building data has been proposed time and again over the last decade [4,39,46]. Unlike conventional data modeling approaches that are limited by the scope of their underlying schemas, these Semantic Web technologies provide an open and common environment for sharing, integrating and linking data from different domains and databases. Semantics can be formally defined with the logic basis of these technologies and shared using web-based mechanisms such as Uniform Resource Identifiers (URIs) and the Hypertext Transfer Protocol (HTTP). The ifcOWL ontology has been developed as a counterpart of the IFC meta model using the Web Ontology Language (OWL) and RDF. The ifcOWL meta model is in the final stages of the standardization process driven by the buildingSMART organization, the most important industry standardization body and forms the foundation for Semantic Web applications for the AEC domain [39]. By transforming IFC building models to RDF data that follows the ifcOWL ontology, using a standard query language such as SPARQL to process them becomes possible [19]. This approach is especially applicable for scenarios in which reasoning can be applied and federated data needs to be processed.

By using plain SPARQL on ifcOWL data, however, some of the aforementioned issues still remain to be addressed. Many query and analysis use cases in the AEC domain are hampered by the complexity of IFC data and many required relationships and properties e.g. property sets, product geometry quantities and spatial and topological relations etc. are difficult to retrieve. In this paper, we use SPARQL as a base query language and propose to extend it with a set of functions specific for querying ifcOWL building data. The motivation is elaborated in section 2. We focus on the official IFC documentation and common BIM requirement checking use cases to define required func-

tions. Some of the use case examples are presented in section 4. The strategy of extending SPARQL functions for domain specific usage has also been employed in other industry domains. For example, the Open Geospatial Consortium (OGC) has standardized GeoSPARQL as a set of vocabularies and functions for geospatial data [41], allowing e.g. to implement spatial queries (e.g. 'within distance', 'touching' etc.). We argue, that the standardization, implementation (e.g. Marmotta, Stardog, Oracle, GraphDB etc.) and industry adoption of GeoSPARQL provides a reasonable indication of the feasibility of a similar approach for spatial data in the AEC industry.

There are currently three major components of the BimSPARQL project presented in this paper: 1) A set of functions modelled as RDF vocabularies that can be used in SPARQL queries (see section 4); 2) A set of query transformation rules to map functions to IFC data structures to make writing queries easier (see section 5); 3) A module for implementing geometry-related functions for deriving implicit information (see section 5). The official IFC specification and real-world BIM requirement checking use cases in the Netherlands and Norway, and some checks that have been implemented in Solibri Model Checker (SMC) [10,45,50,51] have been used to drive the development of the proposed and implemented functionality. The links to the vocabularies, transformation rules and source code repository of the prototypical reference implementation is provided in Appendix A.

The extended functions in this research do not require extensions for the grammar of SPARQL. With SPARQL as a common interface language, extended functions can be used to query building data alone or combined with data from other sources, which in turn may have their own domain specific functions (Fig. 1). We believe that this is a generic approach that is usable in many different use cases, including e.g. multi-model collaboration, quantity take-off and cost estimation, requirement and code compliance checking etc.. As a W3C standard, SPARQL has been widely implemented by a plethora of RDF Application Programming Interfaces (APIs) and databases, and there are many of them support extending functions (e.g. see section 3.3 and section 5), hence can be used as base environments for implementing extended functions.

This paper is structured as follows: In section 2, the background of IFC and ifcOWL is briefly introduced and the motivation of this research is elaborated. In section 3, an overview of related research is provided. The proposed functional extensions for SPARQL are

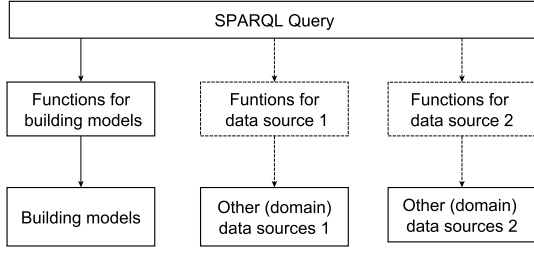


Fig. 1. SPARQL query with domain specific functional extensions

introduced and classified in section 4, followed by example use cases. In section 5, implementation methods are described and a prototype is presented. In section 6, real use cases and building models are used to evaluate this prototype and demonstrate the value of this approach in comparison with SPARQL and exiting work. In section 7, an extended example is presented to show the extensibility of this method. A discussion about added value, limitations and further work concludes this paper.

2. Background and motivation

In the last two decades, the IFC standard has been developed and maintained by buildingSMART as a standard data model for data exchanges between heterogeneous applications in the AEC/FM sector [10, 15]. The IFC schema is specified using the EXPRESS modeling language [24], while its instances are usually serialized in IFC STEP File format [25]. The comprehensiveness of the AEC domain makes IFC one of the largest EXPRESS-based data models across engineering industries. It provides a wide range of constructs for modeling building-related information. For example, one of the most recent versions, IFC4_ADD1, defines 768 entities and 1480 attributes on the schema level [10]. IFC has also provided a few mechanisms to extend semantics in the instance level including e.g. common property sets and external standard classification references. At the same time however, this extensibility constitutes additional difficulties for the implementation of the IFC model in software. On the other hand, the information required in the AEC industry is still much more than all available concepts defined in the current IFC. Therefore, a large amount of information is informally or implicitly represented and usually causes redundancies and ambiguities in IFC instance models [52]. As an object-oriented data model, IFC structures data mainly for the purpose of data ex-

change rather than for the understanding of the knowledge domain, and information is usually represented using relatively complex structures. All these issues have brought about difficulties regarding data query and management of IFC instance data.

Converting the IFC schema and its instances to OWL and RDF was firstly proposed and implemented in [4] to facilitate use cases of data partition, data query and knowledge reasoning. It has been further developed by the buildingSMART Linked Data Working Group (LDWG) and has been specified as candidate standard status in 2015 [39]. Using inferencing and reasoning capabilities of RDF(S) and OWL, practical data processing scenarios in the building industry can be addressed with off-the-shelf algorithms and tools that would required custom tailored tools using STEP-based modeling. For example, a simple data validation use case requires that every building element should be associated with a building storey, can be implemented without hardcoding procedural validators [45,58]. The relationship between a building element and the related building storey can be defined using an instance of *IfcRelContainedInSpatialStructure*, which is an objectified relationship defined in IFC. Provided that the building model is represented in the standardized ifcOWL, the query provided in Listing 1 can retrieve building elements which do not have this spatial containment relationship using common SPARQL implementations.

```

SELECT ?e
WHERE {
  ?e a ifc:IfcBuildingElement .
  FILTER NOT EXISTS {
    ?r ifc:relatedElements ?e .
    ?r a ifc:IfcRelContainedInSpatialStructure .
    ?r ifc:relatingStructure ?storey .
    ?storey a ifc:IfcBuildingStorey .
  }
}
  
```

Listing 1: Query to retrieve building elements which are not contained in a building storey. The query result can be used to check the spatial containment relationship for every building element. ²

²In this paper, all properties defined in ifcOWL are abbreviated to compact format in query listings e.g. `ifc:relatedElements` is used to represent the standardized `ifc:relatedElements_IfcRelContainedInSpatialStructure`. Another simplification is that all the queries in this paper assume that they are under RDFS entailment, hence in this case all the instances of `IfcBuildingElement` subtypes are visited by the query.

As RDF and Linked Data have received increasing attention in the AEC industry, it makes sense to use SPARQL as a common language to process federated data sources instead of developing custom domain specific languages. Common, real-world instance model query scenarios that can be implemented using the current SPARQL specification include:

- All building objects should be tagged with NLsfb classification code, which is a building product classification system used in Netherlands.
- The type and thickness of walls can only be modelled according to the valid combinations provided in an external table X.
- Retrieve all geo-locations of companies which produce the materials used in the walls placed in space X.

All these tasks not only need to query building models captured in e.g. IFC, but also require data from other sources. We argue that they can be more easily implemented with RDF and SPARQL technologies without relying on proprietary systems.

The conversion from IFC instances to ifcOWL RDF data is a straightforward process, and the data structures in IFC instances are reflected in the output RDF data [39]. Since standard SPARQL queries are only processed by matching the data graph patterns in RDF, the resulting queries are usually more complex than the high-level abstractions provided in use cases. For example, in the query case of Listing 1, it is better to have a shortcut relationship between a building element and a storey rather than the objectified solution of the regular schema. There are many commonly used structures that can be simplified all over the IFC meta model to simplify query and make properties and relationships closer to the understanding of knowledge domains.

Another problem that motivates this research and development work is that SPARQL can hardly retrieve useful information in scenarios where geometric computations and spatial reasoning is needed. Geometry data usually constitutes the largest sections in building models (see Table 10) and contains large amounts of information that currently can only be interpreted by human domain end users. Although the IFC meta-model provides many ways to explicitly model geometry-related properties and topological relationships (e.g. property sets and explicit relationships such as the *IfcRelContainedInSpatialStructure* relationship used in Listing 1), they are not mandatory and not always reliable due to lack of rigidity in the IFC meta model and the ad-hoc nature of de-

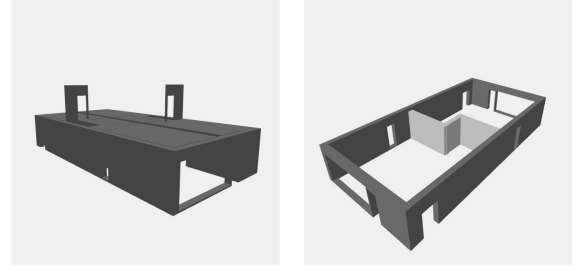


Fig. 2. A model that has incorrect semantic information with respect to its geometric data. The left one shows two walls which are stated as "contained in" (using *IfcRelContainedInSpatialStructure*) in a storey are actually located on the storey above. The second one shows three walls (the light grey ones) which are labelled as "is external" are actually internal.

sign processes in the AEC domain. In practice, IFC building models often miss required semantic relationships and properties or contain incorrect or inconsistent information. Figure 2 shows two examples of inconsistencies between semantic relationships and geometric representations in real building models. Directly deriving information from geometric representations of building models provides another option to enrich data and ensure consistency. Furthermore, much geometry-related information is impractical or impossible to be explicitly provided in IFC data. For example, there are specific topological relationships such as the "touching" relationship between the bottom surface of a wall to the upper surface of the floor slab (see Listing 8) [50], or properties such as distances between elements (see Listing 7).

Across the different use cases analyzed in the context of the research presented here [50,45,10,51], there are many commonly used concepts that are frequently reused. Using the query in Listing 1 as an example, the spatial containment relationship is required in data validation use cases, and is also important in many cases including e.g. cost estimation. By wrapping them as functions used in a standard language (see section 4), we are able to reuse them in many different cases.

3. Related work

3.1. BIM query techniques

Many past developments have been aimed at the query and analysis of IFC instance data. Some commercial platforms such as Solibri Model Checker (SMC) provide functions for querying IFC data [51].

However, the semantics of query functions in these proprietary systems are not transparent and the usage of them is limited by the user interfaces provided to end users.

Standard query languages have been used for querying IFC data in the past. As the IFC data model is based on EXPRESS, the standard languages EXPRESS and EXPRESS-X can be used for queries [24]. However, the EXPRESS language family has not gained popularity outside the STEP initiative in either engineering or software development communities, and there is a very limited set of tools to support them. Some other attempts have used the generic Structured Query Language (SQL) to query IFC data that has been mapped into relational databases [28,30]. These attempts either have severe performance and scalability issues due to the vast amount of tables, or are not intuitive enough for end users.

BimQL is among the first implemented and open source domain specific query language for querying IFC data [34]. It is implemented in the open source bimservers.org platform [4]. It provides create, retrieve, update and delete (CRUD) functionalities to manipulate IFC data. Besides using concepts in the IFC schema, BimQL also provides a few shortcut functions for handling common use cases such as deriving information from common modeling constructs in the IFC model referred to as property sets and quantity sets. However, these functions are very limited and BimQL has not been further developed.

Geometry and spatial information in building models is especially focused by a spatial query language introduced in [8]. This approach is further developed as a query language named QL4BIM for querying IFC data [11]. It has provided a few topological and spatial operators and use R-Tree [17] spatial indexes to optimize query performance.

There are also query languages tailored for specific use cases such as building code compliance checking. The Building Environment Rule and Analysis (BERA) Language is a domain-specific language dedicated to evaluate building circulation and spatial programs [32]. For this purpose, it has defined an internal data model containing a small subset of IFC with related concepts such as floor, space and door, etc. Path-finding algorithms are developed to generate circulation routes between spaces. As a language, however, BERA has limited expressive power and only supports some specific cases on building circulation rules. BIM Rule Language (BimRL) is a more recent research project [13]. It is a domain specific query lan-

guage designed to facilitate accessing information for use cases of regulatory compliance checking. BimRL has provided a suite of components including a simplified data schema and a light-weight geometry engine. IFC building models are loaded through an Extract-Transform-Load (ETL) process into data warehouse. The language has an SQL-like syntax to check building models in terms of the defined data schema and implemented functions. It is currently implemented based on a relational database.

The above technologies have provided inspiring domain specific algorithms for querying building data. Currently, however, no query language has been standardized or widely adopted by the research community and AEC industry. We argue that this might be because these technologies are limited by the closed conventional data modeling approaches that are not sustainable in the AEC domain, which continuously needs changes, extensions and customizations according to different contexts and use cases. All these domain specific BIM query languages are designed based on fixed internal data models (usually an IFC equivalent or a simplified subset of it) and additional functions are hard-wired on top of them. Although some of them have provided programming interfaces for further extensions, the development work is usually limited by the data captured in its internal data model.

3.2. Applying Semantic Web technologies for querying BIM models

In recent years, Semantic Web and Linked Data technologies have received increasingly more attention as a knowledge modeling approach in the AEC industry and a number of research prototypes have been developed. A recent and comprehensive overview of them is provided in [40]. Here, we only briefly describe cases related to data query and knowledge reasoning tasks.

Regarding data query for use cases in the AEC domain, one of the early examples is described in [56]. Conformance constraints are interpreted and formalized as SPARQL queries in that paper. A similar method is developed in [9], which has introduced a semi-automatic process to transform regulatory texts to SPARQL queries. A limitation of both efforts is that they mainly focus on formalizing building regulations into a query language without specifying on how to map the used terminologies to building data models.

A number of researchers have applied Semantic Web technologies in different sub-domains in the con-

text of the AEC industry to facilitate knowledge modeling and rule checking. In [36], a remarkable approach for facilitating regulatory compliance checking has been introduced based on N3Logic and EYE reasoning engine [5], and a test case of an acoustic performance checking is presented. In [33], an OWL ontology has been used for reasoning tasks in cost estimation cases. There are also cases regarding energy management and simulation, construction management, and job hazard analysis etc. [2,57,60]. All these examples have proved that different knowledge reasoning tasks in the AEC industry can be facilitated by properly using Semantic Web technologies.

Currently however, a systematic way to query data from building models using Semantic Web technologies is still missing. One of the possible reasons is that an authorized and stable standard ifcOWL ontology has only been established very recently and its adoption in suitable use cases will likely take a few more years. The most similar work that has overlaps with this research are the IfcWoD and SimpleBIM [16,38] ontologies. They both attempt to transform ifcOWL data to a more compact graph to ease query and improve runtime performance. The difference is that they mainly focus on developing a standard ontology as an alternative to ifcOWL to simplify the data graph, while this research is a framework that mainly considers the query functions with respect to semantics in common use cases and further extensions of them. A major enhancement of the approach introduced here is that functions related to geometry data are provided. To our knowledge, it is the first time to combine analyzing IFC geometry data with rule-based reasoning technologies.

3.3. Functional extensions of SPARQL

Extending SPARQL with additional functions has been proposed and implemented in other fields. The most inspiring ones are geospatial and geographical domains as they share many requirements, concepts and processes with the AEC industry. The stSPARQL in Strabon and the GeoSPARQL standard from Open Geospatial Consortium (OGC) have specified many topological and geospatial functions for 2D geometry data [31,41]. They have been implemented by spatial database systems including Strabon, Parliament and uSeekM [18]. Some other RDF APIs and triple stores like the Apache Jena framework, Allegrograph and OpenLink Virtuoso have also implemented some geospatial functions. To our knowledge, these vocab-

ularies and functions developed in the Semantic Web world have mainly considered 2D geometry and cannot be directly reused for building models.

The AEC industry also has significant differences to e.g. the geospatial field. There are many disciplines and use cases in different contexts, in which the amounts of required properties and relationships are almost unlimited. There are much more sophisticated reasoning tasks related to 3D geometry. Therefore, the systems needed in the AEC domain must go beyond a fixed set of vocabularies but should rather provide a flexible framework that can relatively easy to reuse and extend functions to process data and adapt with different situations.

From the implementation perspective, there are many technologies can be used to extend functions for SPARQL. Besides existing open source and commercial platforms (e.g. Apache Jena, OpenLink Virtuoso and Allegrograph) that support customizing functions by coding them with full fledged programming languages, there are some technologies that provide more transparent and portable methods for extending functions. For example, SPARQL Inferencing Notation (SPIN) can be used to define and execute functions by issuing SPARQL queries. A meta vocabulary is provided by SPIN to serialize SPARQL queries into RDF graphs to maintain implemented functions (see section 5). The VOLT proxy provides a similar method that utilizes SPARQL fragments and graph patterns to define functions [44]. It has been applied on some geospatial cases and a plugin to include functions for spatial computation is provided based on the PostGIS API. Recently, an approach is presented in [12] to define functions by extending Triple Pattern Fragments (TPF) [53] on the client side, hence extended functions are compatible with any SPARQL server. As showed in [12], it however might have issues regarding performance and data traffics since additional functions are computed in web browsers and raw data needs to be retrieved to the client side. All these approaches can potentially be undertaken for implementing extended SPARQL functions for querying IFC building models and data in the AEC domain.

4. Vocabularies

Building data captured by the IFC data model is the focus for developing functions. The IFC documentation and requirement checking use cases from the Dutch Rgd BIM Norm, the Norwegian Statsbygg

BIM Manual and some checks that have been implemented in the Solibri Model Checker (SMC) are reviewed to determine the structure of needed vocabularies [10,45,50,51]. Most of the referenced cases are BIM data quality validation requirements, which are associated with the IFC data model and are the most fundamental and commonly-used requirement checking cases. From reviewing the above sources, we have extracted many properties and relationships that are required in use cases (see section 4.1, 4.2, 4.3 and 4.4). The implemented functions are wrappers of modular low-level code to derive such information and to coherently use them in different scenarios. Due to the complexity of the AEC industry however, it is not possible for a single organization to list all required functions for all common task scenarios. Instead, they are classified based on required data inputs from IFC building models since they are very much related to further implementations and extensions (see section 5).

Information in IFC-based building models can be roughly grouped into 1) domain semantics that are usually explicitly represented by e.g. object types, relationships, and properties, and 2) geometric data, which is a low-level technical description captured by geometry objects associated with *IfcProduct* instances. Due to the lack of support for parametric geometry description on the levels of the meta model and the implementation, these two kinds of information are almost independent from each other. In fact, building models in real practices often contain information that is inconsistent between these two subsets [48] (also see Figure 2). We thus argue that query functions should be categorized to identify which subsets of the model are used to derive data from. As shown in Figure 3 and listed in Table 1, the proposed domain vocabularies are classified into four groups to derive data from these two subsets of either geometric or non-geometric information in IFC models. Sections 4.1 and 4.2 describe functions used to extract information only from domain semantic subset of models, while sections 4.3 and 4.4 describe functions to mainly analyse geometric aspects. Besides these four vocabularies that are defined for building objects, we also propose a vocabulary in section 4.5 to materialize and process geometry data. It is considered as an additional lower level layer independent from domain information and can provide additional functions for some use cases e.g. the example in Listing 8. For each category and subcategory, some function examples are provided to show how to apply them on an ifcOWL instance data set and query examples are provided to demonstrate a use case.

There are generally two ways to extend SPARQL with domain specific functionality. The first method is to add operators in expressions (e.g. *FILTER* expression). The second one is to define a function as an RDF property, which is known as a *computed property* or *property function* to be used in triple patterns to generate or evaluate bindings based on its bound subject and object [54]. The differences are: 1) a property function is also an RDF property that can have domain(s) and range(s); 2) a property function can generate new bindings for triple patterns beyond simply computing values based on inputs. The syntactic sugar of using RDF collections in triple patterns also provide the possibility for a property function to have multiple inputs and outputs (see an example in Listing 7). In the research presented in this paper, most of the extended functions are defined as property functions. We argue, that they are more flexible and intuitive and can potentially be materialized into RDF graphs for specific applications in order to improve runtime performance [37]. Functions are modelled as RDF vocabularies with their respective URIs. Due to the flexibility and openness of the RDF technology, additional vocabularies can always be added.

4.1. Functions for schema level semantics

Functions in this group are defined to wrap commonly used structures specified on the IFC schema level. We model these functions mainly from the *fundamental concepts and assumptions* specified in the official IFC documentation [10]. These *fundamental concepts* describe recommended and commonly used structures in IFC instances as the general guideline for usage and implementation of IFC. Each of the fundamental concepts defines how a domain concept or relationship should be represented in IFC. Many of them have relatively complex structures to represent semantics. By reviewing these fundamental concepts and comparing them with use cases, shortcuts can be constructed to simplify writing queries and adapt to the high level abstractions in the AEC domain. They are defined for the following situations.

The most basic functions are related to objectified relationships. Many relationships in IFC data are realized by objectified relationships that are instances of *IfcRelationship* subtypes. An example is *IfcRelContainedInSpatialStructure*, which is used in Listing 1. Most of these objectified relationships and their usage are described by the *fundamental concepts* in IFC documentation. In general, each of the objectified re-

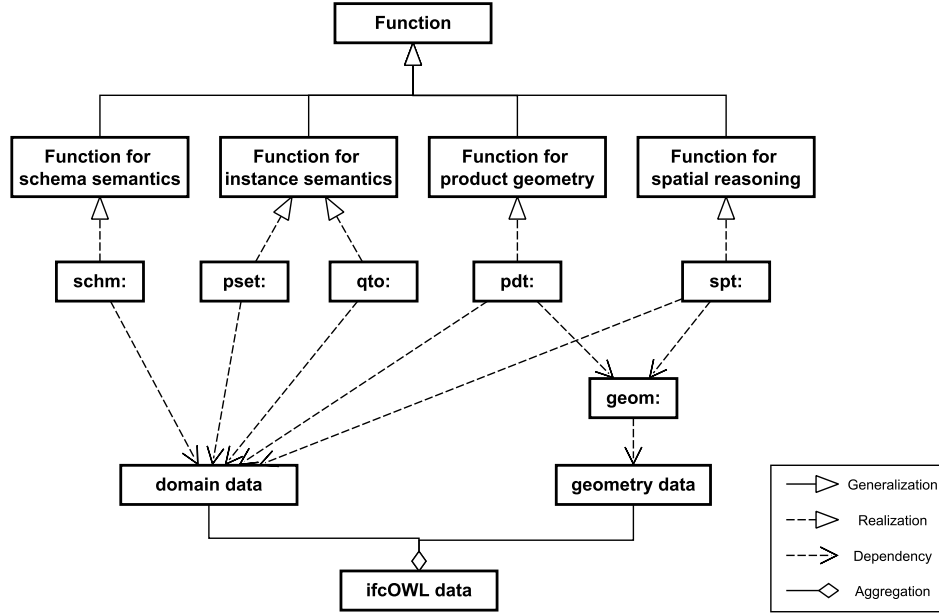


Fig. 3. Conceptual relationships between vocabularies and IFC data

Table 1
Vocabulary prefixes used in this paper and descriptions

Prefix	Description
<code>schm:</code>	Shortcut properties and relationships for IFC schema level semantics (see section 4.1)
<code>pset:</code>	Short cut properties for instance level property sets (see section 4.2)
<code>qto:</code>	Shortcut properties for instance level quantity sets (see section 4.2)
<code>pdt:</code>	Properties for single product based on geometry data (see section 4.3)
<code>spt:</code>	Properties and relationships based on geometry data of multiple products (see section 4.4)
<code>geom:</code>	Lower level geometry library for materializing geometry data and computations on geometry objects (see section 4.5)

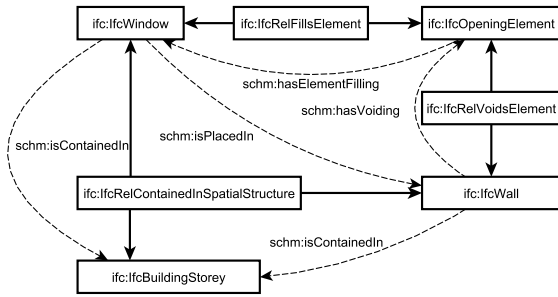


Fig. 4. Example of shortcut functions for schema level semantics

relationships can be used to associate an object with another object or a set of objects. For example, an *IfcRelContainedInSpatialStructure* can be used to associate an *IfcSpatialElement* (e.g. storey, space) with

a set of *IfcElement* instances (e.g. wall, door) to define a spatial containment relationship. In the current vocabulary, functions are defined as shortcuts to wrap such structures and create direct relationships between the objects that are associated. For example, the function *schm:isContainedIn* is created to retrieve the relationship between an *IfcElement* and the containing *IfcSpatialElement* instance (see Figure 4). With the same approach, functions are created for all the *fundamental concepts* which describe semantic structures containing *IfcRelationship* subtypes (see another example *schm:hasSpaceBoundary* in Table 2). This type of shortcuts are also proposed in [16] and [38].

Another requirement is that some relationships need additional specification or generalization. For example, the spatial composition relationship between spa-

tial objects (e.g. site, building, space) is semantically different from the aggregation relationship between building elements (e.g. wall, slab, stair). The former one only represents a hierarchical spatial relationship, while the latter one implies geometry compositional relationship. In IFC, however, they are represented using the same structure (*IfcRelAggregates*). These two structures are defined as two different functions (see one of them *schem:isDecomposedByElement* in Table 2). On the contrary, sometimes more generalized relationships are required for different structures. A typical example is the relationship of material association. There are several means to associate a material with a building object (e.g. single material, layered material), while in many use cases, it requires direct relationship between an object and its associated material. In this case, besides functions for each different structures, an additional function is created to retrieve a direct relationship between an object and its associated material regardless of which representation it is taken (see *schem:hasMaterial* in Table 2).

The third situation is functions for additional shortcuts. They are defined only based on experiences and referenced use cases. A typical example is the relationship between a filling element (e.g. doors, windows) and a voided element (e.g. walls that have openings). If we need to assert such relationship, it is realized in IFC with two objectified relationships and an opening element as illustrated in Figure 4. As such relationship is frequently required, a function is created as a direct relationship between the filling element and voided element (see Figure 4 and Listing 2).

```
SELECT ?window ?wall
WHERE{
  ?window a ifc:IfcWindow .
  ?window schem:isPlacedIn ?wall .
  ?wall a ifc:IfcWall .
  FILTER NOT EXISTS {
    ?wall schem:isContainedIn ?storey .
    ?window schem:isContainedIn ?storey .
    ?storey a ifc:IfcBuildingStorey .
  }
}
```

Listing 2: Query to retrieve pairs of a window and a wall, with the condition that the window is placed in the wall but they are not contained in the same storey.

Following these approaches, over 40 relationships are currently wrapped as functions (see Appendix A). Some frequently used examples are listed in Table 2. Listing 2 shows an example query to apply two func-

Table 2
Example functions for schema level semantics

Function	Description
<i>schem:hasType</i>	Generates or evaluates a relationship between an object occurrence and its type object
<i>schem:hasMaterial</i>	Generates or evaluates a relationship an object with its associated material instances regardless of which structures are taken for associating materials in IFC
<i>schem:hasSpaceBoundary</i>	Generates or evaluates a relationship between a space with its boundary elements (e.g. wall, door or virtual boundary)
<i>schem:isDecomposedByElement</i>	Generates or evaluates a relationship between an element and its child elements

tions for a use case from Statsbygg BIM Manual [50], which requires to check whether every window and the wall it is placed in are contained in the same building storey. This query uses the functions *schem:isPlacedIn* and *schem:isContainedIn*. A comparison with a query using plain SPARQL to realize this use case is presented in section 6.

4.2. Functions for instance level semantics

Functions in this group are provided to represent IFC instance level semantics. As mentioned in section 2, IFC instances can be semantically extended by property sets and quantity sets. These extended properties are modelled as instances of *IfcProperty* or *IfcElementQuantity* in IFC models, which are associated with *IfcObject* instances using certain structures. For example, Figure 5 illustrates two common structures for associating *IfcProperty* with *IfcObject* [10]. An extended property that is modelled as an instance of *IfcProperty* with a related *IfcPropertySet* is associated with an *IfcObject* through either an *IfcRelDefinesByProperties* or an *IfcTypeObject*, which in turn is associated with the *IfcObject* through an *IfcRelDefinesByType*. The semantics of extended properties are identified by their names defined in external documentations. A property which is modelled using the former structure overrides a property modelled using the latter one if they have the same name.

This structure leads to complex declarations in SPARQL even for simple use cases. In this research, shortcut functions are defined to directly connect ob-

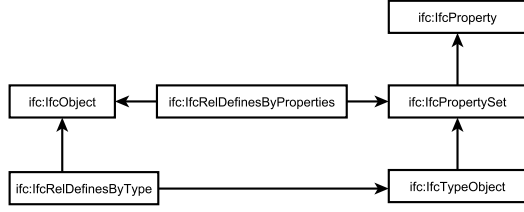


Fig. 5. Two common structures for associating *IfcProperty* with *IfcObject*

jects (*IfcObject* instances) with property values instead of using complex structures in IFC instances for writing queries. These functions are identified with prefixes `pset:` and `qto:` for property sets and quantity sets respectively. A typical example is illustrated in Figure 6, where a wall that has a "LoadBearing" property is represented as an *IfcWall* associated with an *IfcProperty* instance in ifcOWL data. A shortcut property `pset:loadBearing` is defined to associate the wall and value of the property instance. All the properties of primary data types (instances of *IfcPropertySingleValue* and *IfcPhysicalSimpleQuantity*) can use the same mechanism to define functions. They are the majority in property sets and quantity sets and are also most frequently required in use cases. In our work, the property sets and quantity sets officially defined by buildingSMART are considered as examples. In total, there are 2519 properties and 257 quantities grouped within 415 property sets and 93 quantity sets in the official IFC 4 documentation [10]. Within them, 1548 properties and 257 quantities have the value range of primary data types and have been defined as functions.

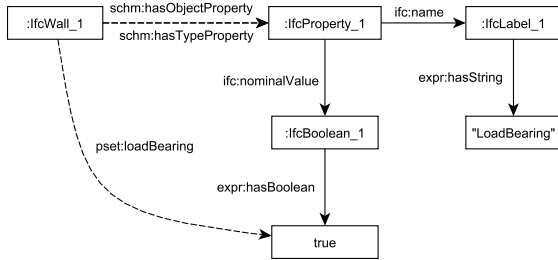


Fig. 6. Example of short cut functions for property sets. The `schm:hasObjectProperty` and `schm:hasTypeProperty` are two short-cut functions defined in the vocabulary `schm:` to wrap the two different structures (see Figure 5) for associating an extended property with an object.

Functions are automatically extracted from the official Ifcdoc document, which is a file in SPF format re-

leased by buildingSMART for storing IFC documentation. Additional, third-party property sets and quantity sets can be extended by processing e.g. simple XML or tabular structures with a trivial tool.

Listing 3 shows a query for a realistic quantity take off example, which is to count the load bearing walls on each building storey. By only using plain SPARQL, a query with the same semantics can also be written but with a much more complex structure (see the comparison in section 6 and Listing 13).

```
SELECT ?storey (COUNT(?wall) AS ?q)
WHERE{
  ?wall a ifc:IfoWall .
  ?wall pset:loadBearing true .
  ?wall schm:isContainedIn ?storey .
  ?storey a ifc:IfoBuildingStorey .
} GROUP BY ?storey
```

Listing 3: Query to count load bearing walls for each building storey.



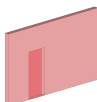
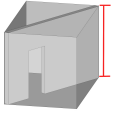



4.3. Functions for product geometry

Functions in this category are introduced to derive properties based on the geometric representations of a single building product. The vocabulary is identified by the prefix `pdt:`. In IFC model instances, geometry data is represented by geometry objects associated with related building products. Large amounts of properties are implied in geometric representations of building products including e.g. height, area, length. Although many of these properties can be represented by property sets and quantity sets (see section 4.2), they are not mandatory and are not always reliable in real building models [48]. In fact, a typical example of BIM requirement checking is to check the consistency between property sets (or quantity sets) and properties derived from geometric representations [50,51].

The IFC meta model offers a number of means to represent geometry for building products. The most common way is the *Body* representation, which defines 3D volumetric shape of products. However, there are many geometry types to describe a *Body* geometry in IFC including e.g. Boundary Representation (Brep), Constructive Solid Geometry (CSG) or Non Uniform Rational B-Splines (NURBS). In our work so far, they are unified as triangulated boundary representation to ease developing analysis algorithms, but can be tailored to different representation forms in future. The 3D geometry representation of a product is either rep-

Table 3

General product geometry function examples that are applicable for all types of product

Function	Illustration	Description
<code>pdt:hasBodyGeometry</code>		Returns the geometry form of a product represented as a WKT literal (see section 4.5). It either retrieves a WKT literal (see section 4.5) to represent a 3D triangulated surface (TIN Z), or a geometry collection (GeometryCollection Z) in WKT.
<code>pdt:hasAABB</code>		Returns the axis-aligned bounding box of a product as a WKT literal (see section 4.5).
<code>pdt:hasMVBB</code>		Returns the oriented minimum volume bounding box of a product as a WKT literal (see section 4.5).
<code>pdt:hasOverallHeight</code>		Returns the height of axis aligned bounding box of a product as a numerical value.
<code>pdt:hasSurface</code>		Returns all plain surfaces of a product. Each of the surfaces is generated as a new binding for the triple pattern which uses this function.
<code>pdt:hasUpperSurface</code>		Returns the upper surface of a product, which is defined as surfaces that have the highest elevation and have normals of nearly (0,0,1), represented as a WKT literal (see section 4.5). A use case of it is shown in Listing 8.
<code>pdt:hasVolume</code>		Returns the volume of the product as a numerical value.

resented by a single triangulated surface, a collection of triangulated surfaces or represented by triangulated surfaces associated with its composing elements.

Based on the triangulated representation, many general geometry properties are derived using existing or simple algorithms (see section 5), including axis-aligned bounding box, oriented minimum volume bounding box, basic dimensions (e.g. height, volume, area of surfaces) and partial geometry (e.g. surfaces facing to certain directions). These properties are defined as general product geometry functions that are applicable for all products which have 3D representations. Table 3 lists examples of them, their 3D show-cases and semantics. Listing 4 shows a use case: compare the height of a wall derived from its geometric representation with its height quantity with a tolerance value [51].

Combined with product types and some common assumptions (e.g. a wall length is greater than the wall

Table 4

Example functions to derive geometry properties for specific product types

Prefix	Description
<code>pdt:hasSpaceArea</code>	Returns the area of bottom surface of a space.
<code>pdt:hasWindowArea</code>	Returns the area of the largest surface of the oriented minimum bounding box of a window.
<code>pdt:hasGrossWallArea</code>	Returns the area of the largest surface of the wall plus area of openings on it.

thickness), many more specific product properties can be retrieved. These properties include some defined examples in Table 4. They can be applied for more domain related use cases such as design assessment. Listing 5 shows an example, which is defined to find out spaces which have too small window-to-floor area ratios. It is a common use case that can be additionally

customized (e.g. add conditions for space types) to validate the design plan according to regulations or programmatic requirements.

```
SELECT ?w
WHERE{
  ?w a ifc:IfcWall .
  ?w pdt:hasOverallHeight ?hg .
  FILTER NOT EXISTS {
    ?w qto:height ?h .
    FILTER(?hg>?h-0.01 && ?hg<?h+0.01)
  }
}
```

Listing 4: Query to retrieve walls that do not have height quantity or have inconsistent information between its height quantity and geometric representation.

```
SELECT ?space ?ratio
WHERE{
  ?space a ifc:IfcSpace .
  ?space pdt:hasSpaceArea ?area .
  {
    SELECT ?space (SUM(?windowArea) AS ?totalWindowArea)
    WHERE{
      ?space schm:hasSpaceBoundary ?w .
      ?w a ifc:IfcWindow .
      ?w pdt:hasWindowArea ?windowArea .
    } GROUP BY ?space }
  BIND ((?totalWindowArea/?area) AS ?ratio)
  FILTER (?ratio<0.3)
}
```

Listing 5: Query to retrieve spaces which have window-to-floor area ratios less than 0.3.

4.4. Functions for spatial reasoning

Functions in this group are provided to derive information related to spatial reasoning, which needs geometric and location data of multiple building products. This vocabulary is identified by the prefix `spt:`. They are additionally classified and described in following sections.

4.4.1. Relationships between products

Functions in this category are used to derive relationships between two products. We have defined some general topological relationships that belong to this group, which are applicable for all building products. They are related to many use cases including e.g. geometric clash detection and quantity take-off. Defined

functions are listed in Table 5 with their counterparts defined in OGC Simple Features and example scenarios for using such functions [21]. Each of these functions retrieve products that have such relationships, or evaluate the relationship between two products. In the GeoSPARQL standard, these topological relationships are defined to process 2D geometry data, while in our cases 3D geometry data is the focus.

Listing 6 shows an example to retrieve walls which intersect with slabs in order to detect clashes between walls and slabs.

Table 5
Functions for relationships between products

Function	Simple counterpart	Use case scenario
spt:touces	touces	Identify connection relationships between building elements
spt:disjoints	disjoints	Evaluate interferences between building elements
spt:intersects	overlaps	Detect clashes between building elements
spt:contains	contains	Identify containment relationships between e.g. space and elements
spt:within	within	Identify containment relationships between e.g. space and elements
spt:equals	equals	Detect duplicate building elements in coordination phases

```
SELECT ?wall
WHERE{
  ?wall a ifc:IfcWall .
  ?slab a ifc:IfcSlab .
  ?wall spt:intersects ?slab .
}
```

Listing 6: Query to retrieve all walls intersect with slabs. The result of query is used to detect clashes between walls and slabs.

4.4.2. Property for groups of products

Functions in this group are used to derive properties for groups of products. Querying the distance between products is a typical example. Many building codes and BIM requirement manuals constrain the minimal, maximal or exact distance between building components, such as interference between building elements, clearance before openings, heights of floors etc. The

exact semantics of the notion "distance" can vary between contexts. We have currently defined the concepts provided in Table 6.

Table 6
Functions as properties for groups of products

Function	Description
spt:distance	Returns the shortest distance between two products in 3D space
spt:distanceZ	Returns the vertical shortest distance between bounding boxes of two products
spt:distanceXY	Returns the shortest distance between the projections of two products on a horizontal plane

An example query is provided in Listing 7 to detect suspended ceilings that are too close to the floor slab and may e.g. interfere mechanical, electrical, and plumbing components (MEP) by selecting ceilings which have the vertical distance shorter than 0.4 meter with floor slab in the above floor [50]. The function *spt:distanceZ* requires two products as the inputs for the computation.

```
SELECT DISTINCT ?ceiling
WHERE{
  ?ceiling a ifc:IcfCovering .
  ?ceiling ifc:predefinedType ifc:CEILING .
  ?ceiling schm:isContainedIn ?storey1 .
  ?storey1 spt:hasUpperStorey ?storey2 .
  ?slab schm:isContainedIn ?storey2 .
  ?slab a ifc:IcfSlab .
  ?slab ifc:predefinedType ifc:FLOOR .
  (?slab ?ceiling) spt:distanceZ ?distance .
  FILTER (?distance<0.4)
}
```

Listing 7: Query to retrieve ceilings that are too close to the floor slabs in the above floor.

4.4.3. Property and relationships based on spatial relationships

In the considered use cases, there are also examples that not only require geometry data of referenced products, but also require to process geometry data of other specific types of related building products. For examples, spatially identifying whether a building storey is located right above another one requires geometry and location data of floor slabs of all the building stories, and retrieving a walking path between two spaces requires geometry data of all the related spaces, obstructions and openings. The exact semantics of these prop-

erties often require knowledge from AEC sub-domains for their specification. We currently only provide two example functions listed in Table 7 for this group. Besides referenced products (building storey and building elements), they both require to process geometry data of floor slabs of all building storeys.

Table 7

Implemented example functions as properties based on spatial relationships

Function	Description
spt:hasUpperStorey	Generates or evaluates bindings between a building storey and the storey right above it
spt:isLocatedInStorey	Generates or evaluates bindings between an element and the building storey which spatially contains it

An example query which uses the function *spt:hasUpperStorey* is shown in Listing 7.

4.5. Geometry library

This vocabulary includes geometry related concepts that are materialized in RDF graphs. They are considered as general geometry concepts that provide additional layers independent from domain information. Similar with GeoSPARQL, we define the *geom:Geometry* as the class for geometry objects. As mentioned in section 4.3, triangulated representations are used to represent *Body* geometry data. As geometry data for a product is usually processed as a whole, Well Known Text (WKT) string literals that have been defined in Simple Feature Access [21] are adopted to keep materialized triples in small size. The geometry data of an element (instances of *IfcElement* subtypes) that is decomposed by other elements is represented by geometry data of its composing elements. Figure 7 illustrates the basic structure for materializing product geometry data. Table 10 lists a comparison between triple count of building models in ifcOWL, geometry subsets of them and the triple count of geometry data represented in this format. Besides the triangulated representations that are by default always materialized, the axis aligned bounding boxes and minimum volume bounding boxes for products are also provided in this vocabulary. In future research, other types of geometry representations can also be extended if they are required.

Another requirement that can be addressed by WKT and this vocabulary is to represent and process tem-

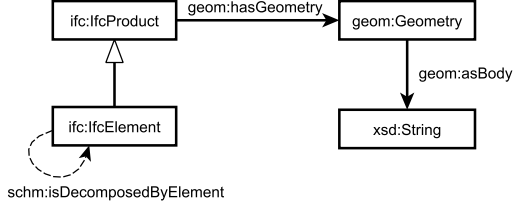


Fig. 7. SPARQL query with domain specific functional extensions

porarily generated geometry data at query runtime. In many tasks, analysis on IFC building models not only requires geometry data of building products, but also needs temporarily defined or derived geometry data. Figure 8 shows some use cases of them. Such geometry can be manually added or automatically derived in query runtime with the WKT literals and expression functions used in e.g. *FILTER* expressions can be defined for additional manipulation on them. An initial set of expression functions for manipulating WKT data are defined. The query example in Listing 8 demonstrates an example of using them. In this example, the bottom surface and upper surface are derived at query runtime (see Table 3) as partial geometries of a wall and a slab, and they are additionally evaluated by the function *geom:touches3D* to identify their topological relationships.

```

SELECT ?wall
WHERE{
  ?wall a ifc:IfcWall .
  FILTER NOT EXISTS{
    ?wall schm:isContainedIn ?storey .
    ?slab a ifc:IfcSlab .
    ?slab schm:isContainedIn ?storey .
    ?slab ifc:predefinedType ifc:FLOOR .
    ?wall pdt:hasBottomSurface ?ws .
    ?slab pdt:hasUpperSurface ?ss .
    FILTER (geom:touches3D(?ws,?ss))
  }
}

```

Listing 8: Query to select all walls which do not have bottom surface touching the upper surface of any floor slab on the same floor

5. A prototype implementation

In our prototype implementation of the proposed functions, we attempt to minimize hardcoding to make defined functions more portable, more transparent for

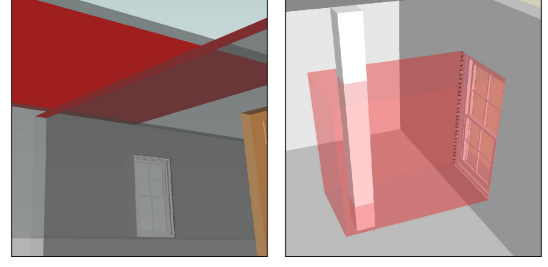


Fig. 8. Use case examples that require temporarily added or generated geometry objects to analyse properties and relationships of building objects: The first one requires "upper surface" and "lower surface" of walls and slabs to evaluate their topological relationships; The second one requires extruded boxes to evaluate clearance in front of windows.

public reviews and easier to be extended by the research and development community. Table 8 lists the current amount of defined and implemented functions.

Table 8
Count of currently defined and implemented functions

Prefix	Property function	Expression function
schm:	46	-
pset:	1548	-
qto:	257	-
pdt:	15	-
spt:	11	-
geom:	-	19

Functions defined in section 4.1 and 4.2 can be implemented by a range of methods including those described in section 3.3 and declarative rule languages like e.g. Semantic Web Rule Language (SWRL) and N3Logic [5,22]. We choose SPIN for the implementation, as it uses SPARQL and already has a few open source implementations which enhance future compatibility. SPIN provides a set of vocabularies to wrap SPARQL queries as functions and allows their cascading use. For example, the function *schm:isContainedIn* in Listing 2 is mapped to ifcOWL with the query in Listing 9. As presented in Listing 10, this function is maintained as an instance of *spin:MagicProperty*, and the query is transformed to RDF and associated with the function using *spin:body* property. The system will trigger the query as a subquery when the function *schm:isContained* is called as the predicate in a triple pattern. In this process, the subject and object of this triple pattern will be passed to *?arg1* and the output (in this case the *?a2*) of the query respectively to generate or evaluate bindings. An advantage of using such method for implementing functions is that devel-

opment work is more portable. For example, this function can run in any environments where SPIN is implemented.

```
SELECT ?a2
WHERE {
  ?a1 ifc:relatedElements ?arg1 .
  ?a1 ifc:relatingStructure ?a2 .
  ?a1 a ifc:IfcRelContainedInSpatialStructure .
}
```

Listing 9: Query that is used in SPIN to map the function *schm:isContainedIn*

```
schm:isContainedIn
rdf:type spin:MagicProperty ;
rdfs:subClassOf spin:MagicProperties ;
rdfs:domain ifc:IfcElement ;
rdfs:range ifc:IfcSpatialStructureElement ;
spin:body
[ rdf:type sp:Select ;
  sp:resultVariables ([ sp:varName "a2" ]) ;
  sp:where
  ([
    sp:object spin:_arg1 ;
    sp:predicate ifc:relatedElements ;
    sp:subject [ sp:varName "a1" ]
  ] [
    sp:object [ sp:varName "a2" ] ;
    sp:predicate ifc:relatingStructure ;
    sp:subject [ sp:varName "a1" ]
  ] [
    sp:object ifc:
      IfcRelContainedInSpatialStructure ;
    sp:predicate rdf:type ;
    sp:subject [ sp:varName "a1" ]
  ])
] ;
spin:constraint
[
  rdf:type spl:Argument ;
  spl:predicate sp:arg1 ;
  spl:valueType rdfs:Resource
] .
```

Listing 10: SPIN listing (Turtle syntax) for the query in Listing 9, which is used in SPIN to register and define the function *schm:isContainedIn*

When dealing with geometry related reasoning tasks, these declarative methods are usually not sufficiently expressive to implement sophisticated and computational intensive algorithms. Geometry data in IFC or ifcOWL is preprocessed and transformed to RDF data represented by the vocabulary described in

section 4.5. Functions described in section 4.3, 4.4 and 4.5 are implemented using low-level procedural programming. Many existing general purpose geometry algorithms and domain specific algorithms can be reused. For example, functions in section 4.4.1 are implemented by computing on triangles of both products to determine their relations, similar with algorithms described in [11]. Table 9 lists the key procedurals and algorithms that are used. They are coded in Java in the current prototype.

Table 9

Procedurals for implementing geometry-related functions and used existing algorithms

procedural	algorithm
WKT IO	SFCGAL library [7]
MVBB (see section 4.3)	Jylanki [27]
volume (see section 4.3)	Zhang and Chen [59]
topology operators (see section 4.4.1)	Daum and Borrmann [11]
distance (see section 4.4.2)	SFCGAL library [7]

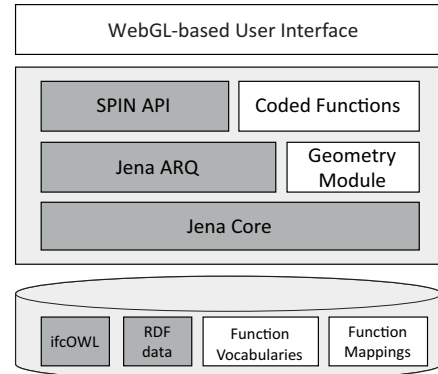


Fig. 9. Implementation architecture (blank blocks are added modules)

The functional extensions introduced here are implemented based on the Open Source Apache Jena framework and SPIN API (see Fig. 9). In this implementation, all the extended functions are processed at query runtime in a backward chaining order. The data flow is illustrated in Figure 10. The ifcOWL instances or IFC files and SPARQL queries are the input of the system. The ifcOWL data or IFC files are preprocessed to generate additional triples that capture geometry data using the vocabulary described in section 4.5 and WKT literals. Depending on the size of ifcOWL files, we can choose to load them into mem-

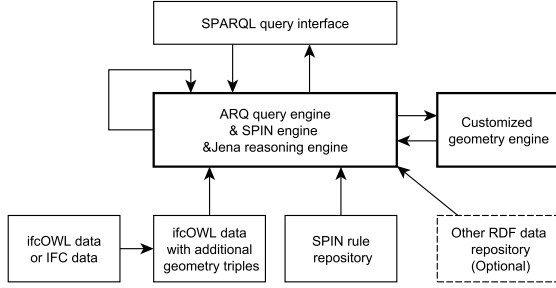


Fig. 10. Data flow of querying and reasoning process

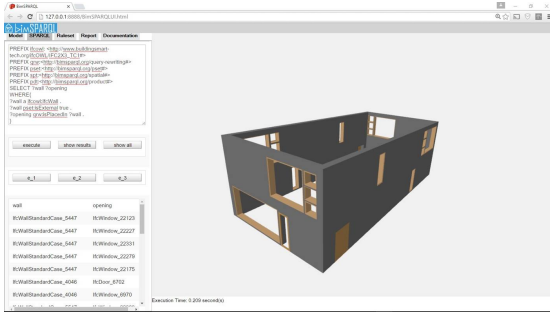


Fig. 11. The Web-based query interface with 3D graphical visualization of this prototype implementation

ory or materialize them into a graph persisted into a Jena TDB triple store. When a property function is referred to during a query execution, a SPIN rule as a subquery or a snippet of programming code to retrieve related values is triggered. Since a SPIN rule is also a SPARQL query that can call extended functions, this process iteratively continues until no functions are left to be called. This process is compatible with other reasoning technologies. For example, in this prototype an Jena RDF Schema (RDFS) reasoner is used underneath of the SPARQL query engine. A prototype Web-based user interface with a 3D visualization environment is implemented to input queries and visualize query results (Figure 11). For example, it highlights retrieved building products in order to report e.g. building products that under certain conditions or violate constraints.

6. Evaluation and comparison

In this section, a validation of the work is presented using three IFC building models employing queries taken from real-world use cases. They are compared with processes realized by standard SPARQL. We also

compare our approach with existing proposals for simplifying ifcOWL data and writing queries. Through this work, we aim to 1) demonstrate the added value as well as the differences of this approach, 2) evaluate this approach and implementation, and 3) provided indicative measurements of query performance.

The models selected for the test are open IFC models commonly used as a reference in literature. They are converted to ifcOWL RDF data and loaded into named graphs persisted in a Jena TDB triple store. Additional WKT geometry triples that capture triangulated boundary representations of building products are generated with the IfcOpenShell package [29]. The size of the different models as well as their specifications are listed in Table 10. For example, the model M1 is 2.25 MB in size in its SPF representation, and the ifcOWL version contains 298,085 triples. 546 additional triples have been generated to capture triangulated boundary representations with the *geom:* vocabulary using WKT literals. All datasets are available at <https://doi.org/10.17605/OSF.IO/V5ENM> (see also Appendix A). In this test, WKT geometry triples are not used to replace the original geometry triples but are simply added and processed along with original ifcOWL models, hence the model M1 that is processed by the query engine contains 298,085 plus 546 triples.

Example queries presented in Listing 2, 3, 4, 5, 6, 7 and 8 in section 4 are used in this evaluation and the results of their execution are presented in this section. Each of the queries addresses a real use case that has been specified e.g. in BIM manuals or implemented as standard model checks in proprietary model checking software tools. They are summarized in Table 11, which lists use case types and requirements. Q1 and Q2 are only related to non-geometric data, while Q3 to Q7 are geometry related.

The hardware used for the evaluation is a mid-range laptop with a Quadcore i7 2670 processor and 4 GB memory allocated for the Java Virtual Machine (JVM). Each of the queries is executed 10 times to derive the average query time.

6.1. Results

Table 12 documents the results and average query execution times for Q1 to Q7 on models M1, M2 and M3. All queries except Q2 are used to address requirement checking use cases by retrieving building objects which violate defined constraints. Thus, in these cases returning zero results means no violation in the building model was detected.

Table 10
Statistics of tested building models M1, M2 and M3.

id	model name	SPF size (MB)	ifcOWL triples	geometry triples in ifcOWL	WKT geometry triples
M1	Duplex_A_20110505.ifc	2.25	298,085	222,212	546
M2	Office_20110811_Combined.ifc	12.8	1,787,763	1,680,645	3,164
M3	091210Med_Dent_Clinic_Combined.ifc	107	14,487,725	12,580,688	15,052

Table 11
Query tested in the evaluation study

id	query body	use case types	description of the query including reference to provenance of real-world use case
Q1	Listing 2	model structure check	Find out windows and walls with the condition that the window is placed in the wall but they belong to different building storeys (Statsbygg, p. 66) [50].
Q2	Listing 3	quantity take-off	Count load bearing walls for each building storey (Solibri example) [51].
Q3	Listing 4	data consistency check	Retrieve walls which do not have the height quantity or height is inconsistent with its geometry representation (Solibri example) [51].
Q4	Listing 5	design check	Find out spaces which have the window-to-floor area ratio smaller than 0.3 (Solibri example) [51].
Q5	Listing 6	design check	Find geometry clashes (intersections) between walls and slabs (Rgd 2.1.6, p. 9) [45].
Q6	Listing 7	design check	Retrieve suspended ceilings that are too close (with the distance less than 0.4 meter) to the floor slabs in the above building storey (Statsbygg 56, p. 35) [50].
Q7	Listing 8	design check	Find out walls that have bottom surfaces not touching upper surfaces of any floor slabs on the same building storey (Statsbygg 41 and 43, p. 30 and 31) [50].

Table 12
Query results and performance of Q1 to Q8 (see Table 11) on M1, M2, M3, M4 (see Table 10).

query	model	triple count (total)	avg. querying (s)	stand. derivation	result count
Q1 (Listing 2)	M1	298,631	0.033	0.053	0
	M2	1,790,927	0.059	0.054	0
	M3	14,502,777	0.110	0.190	31
Q2 (Listing 3)	M1	298,631	0.169	0.033	1
	M2	1,790,927	1.437	0.062	1
	M3	14,502,777	1.610	0.221	1
Q3 (Listing 4)	M1	298,631	0.023	0.00064	49
	M2	1,790,927	0.345	0.0051	495
	M3	14,502,777	0.679	0.0079	750
Q4 (Listing 5)	M1	298,631	0.250	0.025	2
	M2	1,790,927	0.067	0.026	0
	M3	14,502,777	2.377	0.672	41
Q5 (Listing 6)	M1	298,631	1.044	0.188	8
	M2	1,790,927	3.720	0.071	6
	M3	14,502,777	35.471	0.460	10
Q6 (Listing 7)	M1	298,631	0.647	0.03	10
	M2	1,790,927	1.127	0.061	0
	M3	14,502,777	37.152	1.276	0
Q7 (Listing 8)	M1	298,631	0.637	0.103	34
	M2	1,790,927	0.678	0.098	495
	M3	14,502,777	32.386	3.769	65

Table 13
Comparison with a procedural using plain SPARQL

query	triple patterns	results M1	time(s) M1	results M2	time(s) M2	results M3	time(s) M3
Q1 (Listing 2)	6	0	0.033	0	0.059	31	0.110
Q1* (Listing 12)	15	0	0.026	0	0.043	31	0.069
Q2 (Listing 3)	4	1	0.169	1	1.437	1	1.610
Q2* (Listing 13)	40	1	0.207	1	1.89	1	1.70

Q1 and Q2 only depend on functions that are implemented based on the SPIN framework, which in turn is dependent on the Jena ARQ query engine, while Q3 to Q8 also depend on additional computations in external Java code. Q3 and Q4 are related to functions in the group introduced in section 4.3 and their query execution time mainly depends on the algorithms used for deriving properties from the geometry data of a single product. For example, in Q3 when the function *pdt:hasOverAllHeight* is called, an axis-aligned bounding box is generated on the fly from the underlying WKT representation to generate an axis-aligned bounding box in order to derive the overall height of a wall. In Q4, the most computationally expensive part is the function *pdt:hasWindowArea*, which needs to compute a minimum volume bounding box for each window object. These processes can be optimized by materializing additional geometry representations for building products. Q5, Q6 and Q7 have relatively longer execution times, especially for the largest model M3. This is expected since these three queries are all related to spatial reasoning functions, which involve geometry data of multiple building products. For example, the current procedural of the function *spt:intersects*, which is used in Q5, needs to compute the topological relationship for each combination of a wall and a slab. This procedure needs to run 750*19 times for the model M3, which contains 750 walls and 19 slabs. This can be optimized further by mechanisms like adding spatial indices to reduce computation time.

6.2. Comparison

We first compare the results with a procedural that only uses SPARQL to query ifcOWL data. The same query environment is set up with the exception that all the extended functions are not activated. By just using SPARQL and ifcOWL data, only the use cases that are addressed by Q1 and Q2 can be realized, hence the comparison is limited in these two queries. Queries with the same semantics of Q1 and Q2 are written in SPARQL and presented in Listing 12 and Listing 13

in Appendix B. They have complex query bodies that contain more triple patterns. They are documented as Q1* and Q2* in Table 13, which also compares them with Q1 and Q2 with respect to triple pattern count in WHERE clauses, query results and average query time. It shows that with significantly simplified query bodies, Q1 and Q2 have the same query results with Q1* and Q2* respectively without sacrificing much performance. This topic is further discussed in section 8.3.

As mentioned in section 3, there have been a few ontologies developed to simplify ifcOWL data including those introduced in [16] and [38]. All these existing efforts have not considered processing geometry data, hence only the use cases addressed by Q1 and Q2 can be addressed. Functions defined in section 4.1 and 4.2 can be compared with those simplified ontologies. The difference is that those existing simplified ontologies tend to preprocess ifcOWL data (or IFC data) and transform it to a more compact data graph and improve query performance, while the approach presented in this paper treats simplified properties and relationships as functions, which are used in query runtime. An advantage of this approach is that simplified queries can run on any ifcOWL data without additional materializations (for functions defined in section 4.1 and 4.2). This provides a more flexible paradigm that users do not have to adopt the entire vocabulary but can reuse a subset of them or extend them to adapt with more specific use cases. If some simplified IFC ontologies are standardized, this approach can also be compatible with them by defining additional mapping rules.

Regarding use cases addressed by Q3 to Q7, to our knowledge there is no open and off-the-shelf query system in the Semantic Web field can be compared with. Some of them might be supported by BIM query languages which support geometry features like those introduced in [11] and [13]. However, we argue that these query languages either have limited expressive power or do not have precisely defined or standardized semantics, while this approach is based on a standard and expressive query language [1,19]. More im-

portantly, with this approach RDF and other Semantic Web technologies can be leveraged to facilitate knowledge reasoning and data integration and partition tasks. With these capabilities, defined functions can more easily be reused and extended for specific applications. An example is presented in section 7.

7. An extended application example

An application example is presented in this section in a regulatory compliance checking scenario that requires to extend case specific functions to query both building models and regulatory data.

The example provided here is taken from the International Building Code (IBC), which is developed by the International Code Council (ICC) and used as a base code standard in United States [23]. This rule example is from *Chapter 7 Fire and Smoke Protection Features*, and is used to check opening areas on external walls to evaluate their fire performance. This example requires to process domain specific semantic data and geometry data in building models and external tabular data defined in the IBC document.

- 705.8.4 *Where both unprotected and protected openings are located in the exterior wall in any story of a building, the total area of openings shall be determined in accordance with the following:*

$$(Ap/ap) + (Au/au) \leq 1 \quad (1)$$

where:

Ap = Actual area of protected openings.

ap = Allowable area of protected openings.

Au = Actual area of unprotected openings.

au = Allowable area of unprotected openings.

Additionally, the allowable opening areas for protected and unprotected openings (ap and au) are determined by the Table 705-8 in IBC that describes their relations with fire separation distance. This table has three columns and twenty-four rows. Table 14 shows one row of it, which defines that when the fire separation distance is between 15 to 20 feet and the opening is unprotected and the space is non-sprinklered, the allowed ratio (au in the equation) between opening area and external wall area is up to 25 percent.

In this example, external wall instances in a dataset have to be checked and analysed to derive related prop-

Table 14

One row of Table 705-8 in International Building Code [23]

Fire separation distance	Degree of opening protection	Allowable area
15 to less than 20	Unprotected, Non-sprinklered	25%

Algorithm 1. procedure for checking rule 705.8.4

- 1: **for** each external wall w **do**
- 2: compute the area ratio between protected openings and gross wall area Ap ;
- 3: compute the area percentage of unprotected openings and gross wall area Au ;
- 4: compute the fire separation distance for the wall fsd ;
- 5: find the sprinkler status of related space sp ;
- 6: find the allowable area of protected and unprotected openings ap and au from Table 705.8 with fsd and sp ;
- 7: check the equation (1) with Ap , Au , ap , au and return result for w ;
- 8: **end for**

erties and relationships. In addition to the data captured in the IFC building data sets, the referenced table in this example can be considered as a small dataset that needs to be processed to derive allowable protected openings and unprotected openings for each wall. It is transformed to the RDF format with the approach described in [55] and processed along with the building model. A general algorithm in a procedural pseudo-code notation is specified in Algorithm 1 to check building models and find out external walls which violate this requirement.

Case specific functions are extended for deriving some of these properties based on provided functions. For example, the value Ap used in Algorithm 1 is specified as the proportion between all the protected windows in the wall and the gross area of the wall. Based on predefined functions and SPIN rules, this function can be extended with the query provided in Listing 14 (see Appendix C). The value fsd of an external wall used in Algorithm 1 is defined as the horizontal distance between the wall and lot line. Using the same method, functions are extended for this case and listed in Table 15. SPIN rules for defining these case specific functions based on ifcOWL and BimSPARQL functions are listed in Appendix C. As the Table 705-8 in IBC is also processed, a function `ibc:allowableArea_T705-8` is also defined to process

this external dataset. With all these functions extended for this case, the query in Listing 10 is used to check the opening area of all external walls.

Table 15
Extended functions for the rule case 705.8.4 in IBC

Function	Description
ibc:hasAp	Retrieves the ratio between all the protected windows in the wall and the gross area of the wall.
ibc:hasAu	Retrieves the ratio between all the unprotected windows in the wall and the gross area of the wall.
ibc:hasFireSeparationDistance	Retrieves the shortest horizontal distance between a wall and lot lines.
ibc:allowableArea_T705-8	Retrieves allowable area (au or ap) from Table705-8 based on fire separation distance and sprinkler protection status.

```

SELECT ?wall
WHERE{
  # find external walls
  ?wall a ifc:IfcWall .
  ?wall pset:isExternal true .
  # compute Ap and Au values
  ?wall ibc:hasAp ?Ap .
  ?wall ibc:hasAu ?Au .
  # compute fire separation distance
  ?wall ibc:hasFireSeparationDistance ?d .
  # find sprinkler status of related space
  ?wall schm:isContainedIn ?storey .
  ?storey pset:sprinklerProtection ?bool .
  # find ap and au values from the table
  BIND (ibc:allowableArea_T705-8(?d,true,?
    bool) AS ?ap) .
  BIND (ibc:allowableArea_T705-8(?d,false,?
    bool) AS ?au) .
  # filter out walls that have issues
  FILTER ((?Ap/?ap+?Au/?au)>1) .
}

```

Listing 11: Query to retrieve external walls which violate the constraint defined in this building code.

As a proof of concept, a building model is created, which contains required building elements and lot line (modelled as an *IfcAnnotation* instance) with related properties. It is a small model that contains 189,778 triples. With all the additional SPIN functions loaded, it is checked using the query in Listing 11. This prototype implementation generates a visualization of the result that is provided in Figure 12.

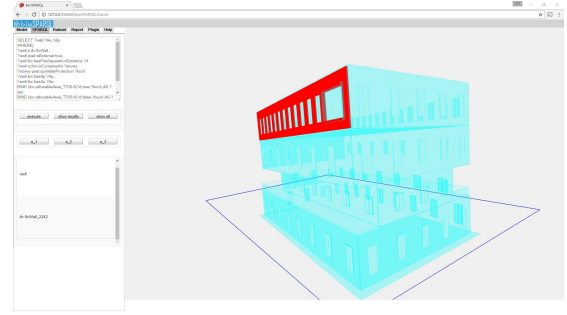


Fig. 12. Snapshot of the query result of the GUI

8. Discussion

It is shown in section 6 that many BIM requirement checking use cases can be addressed by using SPARQL with these functions, which either simplify queries or enhance query abilities. In section 7, we also demonstrate an example to discuss the possibility of extending this framework for specific applications, where more specific functions are required and additional data needs to be processed. Some issues of the work presented here are discussed in this section.

8.1. Flexibility and portability

In comparison with domain specific query languages that are developed from scratch, the approach introduced in this paper leverages Semantic Web technologies and existing implementations to provide a more interoperable, modular and flexible mechanism to extend functionality in order to address a wide range of use cases for information extraction and validation of the AEC industry. As shown in section 7, query functions for specific use cases can be extended by adding additional declarative rules based on procedural functionality. They are modular and flexible to adapt to the various possible forms to present facts in IFC data sets. For example, a protected opening in another building case, created by another author using a different BIM authoring tool might be different from how it is defined in Listing 14. It is easier to change or replace this rule without affecting other rules. External, linked datasets can be addressed using the same technology as long as they are captured as RDF or provide SPARQL endpoint services [6,20].

Declarative methods can also enable more portable implementations for functions. As many functions are defined using SPIN rules, they can be reused by query environments which have implemented SPIN (e.g. Topbraid SPIN API or Eclipse RDF4J) and potentially

be reused by those which have implemented SPARQL. The main issue that affect portability here is functions implemented by procedural programming, which still needs geometry libraries to be integrated.

8.2. Coverage

The full list of implemented functions are published in the link of Appendix A. They are defined based on referenced BIM requirement checking use cases. There are various use cases in the AEC industry and almost unlimited properties and relationships are required. IFC also provides rich methods to represent information to adapt with different contexts and projects. As stated in section 4, it is not our aim to provide a complete set of functions, but to suggest a bottom-up approach to define modular functions and then gradually extend to cover more use cases. In this approach, each function should be considered as a module to retrieve a view from IFC building models. A general classification of them is provided as a framework for further extending functions and a set of functions are provided as foundational examples for this approach.

Functions introduced in section 4.1 and 4.2 cover all the commonly used semantic structures and all the simple data properties and quantities defined in the official IFC documentation. In real practices, these two groups of functions can be extended according to various application concepts in AEC sub-domains and third-party property sets and quantity sets. Functions introduced in section 4.3 and 4.4 mainly focus on triangulated boundary representation, which is a fundamental geometric representation that can be used to represent any 3D physical shapes. A set of general geometry and spatial reasoning functions that are applicable for all building products and some example functions related to specific product types are defined. There are indeed use cases that require particular geometry forms (e.g. deriving the flange thickness for a I-shape beam requires parametric I-shape profile objects), they can be extended by providing multiple representations for specific products.

From the perspective of use cases, a current limitation of this approach is related to requirements of instantiating resources with additional triples based on procedural computations. For example, identifying the shortest path between two rooms usually needs to instantiate a path object, which might have geometric representations and relationships with e.g. passed spaces and doors. Even with procedural coding, extended functions are not suitable to create such addi-

tional data graphs in query runtime as they have side effect for the original data. WKT literals might be used to represent additional geometry objects in query time, but more investigations are still required to properly adapt this type of query functions in an RDF and Semantic Web environment.

8.3. Query performance

At present optimizing query performance is not in the main focus of the research presented here. Query performance depends on the implementation of used technologies and geometry analysis algorithms that are used. The prototype implementation has used the SPIN framework, which is based on the Jena ARQ query engine and Jena TDB triple store. In section 6, it is shown that for some cases, simplified queries can have similar performance with equivalent plain SPARQL queries. This implementation method has also taken part in a performance benchmark with comparisons to other rule languages [37]. That research shows that this implementation method is a reliable approach, but there is still room for optimizing its performance in comparison with some commercial databases like Stardog. In the current SPIN framework, when a function is called in a triple pattern, it is considered as a separate query that is executed based on assigned arguments and then joints with the temporary results of outer query. It lacks a query rewriting mechanism to flatten queries and preferentially execute the most selective triple patterns considering all the triple patterns defined in functions.

As shown in section 6, the current performance short-coming can be mainly attributed to geometry-related functions, especially spatial reasoning related functions. With a plain RDF triple store like Jena TDB without additional optimization mechanisms, spatial reasoning functions have relatively long running time. In future developments, this process can be optimized by e.g. integrating spatial indices and caching mechanisms, which have been proved to have significant impact for spatial reasoning in the geospatial domain [35]. This might lead to creating specialized databases in the future.

As RDF graphs are flexible, another direction for optimizing performance might be to materialize required triples into RDF graphs for specific applications. As described in section 6, besides ifcOWL data only the triangulated boundary representation of products are currently materialized and all the functions are processed at query runtime. If some related function are frequently required for specific applications, it is

recommended to materialize them as properties. The effect of materialization has been discussed in [?]. Since it is usually a trade-off between storing and computing data, a dynamic approach to automatically materialize triples with regards of use cases, query performance and storage cost needs additional investigation and future research.

9. Conclusion

This research provides a general framework to define and extend functions for querying IFC-based building data. A set of functions are classified and introduced and two different approaches are used to implement them. The work presented here should be regarded as a general framework and proof of concept for a modular, scaleable approach to address the large amounts of domain specific query requirements in the AEC domain. As more and more data is represented by RDF and Linked Data technologies, this approach has considerable advantages over the current practices to process building related data in proprietary information silos using one-of-a-kind island solutions.

The links to the vocabularies, transformation rules and source code repository of the prototypical reference implementation is provided in Appendix A. As discussed in section 8, the current performance shortcomings of the solutions introduced here are mainly related to the implementation issues. More efficient algorithms e.g. to preprocess datasets according to spatial indices need to be developed. In future research, more use cases should be investigated and implemented to gradually extend the functionality for specific subdomains in AEC industry and to combine data from different sources. From a technical perspective, optimization for query performance is necessary as it is important for the applications of this approach.

References

- [1] R. Angles and C. Gutierrez, (2008). The expressive power of SPARQL. The Semantic Web-ISWC 2008, pp.114-129.
- [2] K. Baumgartel, M. Kadolsky and R.J. Scherer, (2014). An ontology framework for improving building energy performance by utilizing energy saving regulations. In Proceedings of European Conference on Product and Process Modelling (pp. 519-526).
- [3] J. Beetz, L. Van Berlo, R. De Laat and P. Van Den Helm, (2010). BIMserver.org-An open source IFC model server. In Proceedings of the CIB-W78 conference.
- [4] J. Beetz, J.P. Van Leeuwen and B. De Vries, (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.
- [5] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler, (2008). N3logic: A logical framework for the world wide web. Theory and Practice of Logic Programming, 8(03), pp.249-269.
- [6] C. Bizer and R. Cyganiak, (2006). D2r server-publishing relational databases on the semantic web. In Poster at the 5th International Semantic Web Conference (pp. 294-309).
- [7] M. Borne, H. Mercier, V. Mora, O. Courtin, (2013). SFCGAL. Available on: <http://sfcgal.org/>.
- [8] A. Borrmann and E. Rank, (2009). Topological analysis of 3D building models using a spatial query language. Advanced Engineering Informatics, 23(4), 370-385.
- [9] K.R. Bouzidi, B. Fies, C. Faron-Zucker, A. Zarli and N.L. Thanh, (2012). Semantic web approach to ease regulation compliance checking in construction industry. future internet, 4(3), pp.830-851.
- [10] buildingSMART International, (2013). Industry Foundation Classes Release 4 (IFC4). Available on: <http://www.buildingsmart-tech.org/ifc/IFC4/final/html/>.
- [11] S. Daum and A. Borrmann, (2014). Processing of topological BIM queries using boundary representation based methods. Advanced Engineering Informatics, 28(4), 272-286.
- [12] C. Debruyne, E. Clinton, D. O'Sullivan, (2017). Client-side Processing of GeoSPARQL Functions with Triple Pattern Fragments. LDOW@WWW 2017.
- [13] J. Dimyadi, W. Solihin, C. Eastman, R. Amor, (2016). Integrating the BIM Rule Language into Compliant Design Audit Processes. Proceedings of 34th CIB W78 conference, October 31st to November 2nd, Brisbane, Australia.
- [14] C. Eastman, (1994). Out of STEP?. Computer-Aided Design, 26(5), pp.338-340.
- [15] C. M. Eastman, P. Teicholz, R. Sacks and K. Liston, (2011). BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors. John Wiley & Sons.
- [16] T. M. Farias de, A. Roxin and C. Nicolle, (2015). IfcWoD, semantically adapting IFC model relations into OWL properties. arXiv preprint arXiv:1511.03897.
- [17] A. Guttman, (1984). R-trees: a dynamic index structure for spatial searching (Vol. 14, No. 2, pp. 47-57). ACM.
- [18] G. Garbis, K. Kyzirakos and M. Koubarakis, (2013). Geographica: A benchmark for geospatial rdf stores (long version). In The Semantic Web-ISWC 2013 (pp. 343-359). Springer Berlin Heidelberg.
- [19] S. Harris, A. Seaborne and E. Prudhommeaux, (2013). SPARQL 1.1 query language. W3C Recommendation.
- [20] T. Heath and C. Bizer, (2011). Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology, 1(1), 1-136.
- [21] J. Herring, (2011). OpenGIS Implementation Standard for Geographic information-Simple feature access-Part 1: Common architecture. OGC Document, 4(21), pp.122-127.
- [22] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, and M. Dean, (2004). SWRL: A semantic web rule language combining OWL and RuleML. W3C Member submission, 21, p.79.

- [23] IBC. (2006). International building code. International Code Council, Inc.(formerly BOCA, ICBO and SBCCI), 4051, pp.60478-5795.
- [24] ISO, (1994). ISO 10303-11: Industrial automation systems and integration-Product data representation and exchange-Part 11: Description methods: The EXPRESS language reference manual. International Organization for Standardization.
- [25] ISO, (2002). ISO 10303-21: Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure. International Organization for Standardization.
- [26] ISO, (2013). ISO 16739:2013 Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries. International Organization for Standardization.
- [27] J. Jylanki, (2015). An Exact Algorithm for Finding Minimum Oriented Bounding Boxes. Available on <https://pdfs.semanticscholar.org/a76f/7da5f8bae7b1fb4e85a65bd3812920c6d142.pdf>. Last access: December 2016.
- [28] H.S. Kang, and G. Lee, (2009). Development of an object-relational IFC server. Proceedings of ICCM/ICCPM, 2009.
- [29] T. Kijnen, (2011). IfcOpenShell. Available on <http://ifcopenshell.org/>.
- [30] A. Kiviniemi, M. Fischer and V. Bazjanac, (2005). Integration of multiple product models: Ifc model servers as a potential solution. In Proc. of the 22nd CIB-W78 Conference on Information Technology in Construction.
- [31] K. Kyzirakos, M. Karpathiotakis and M. Koubarakis, (2012). Strabon: a semantic geospatial DBMS. In International Semantic Web Conference (pp. 295-311). Springer Berlin Heidelberg.
- [32] J.K. Lee, C.M. Eastman and Y.C. Lee, (2015). Implementation of a BIM domain-specific language for the building environment rule and analysis. Journal of Intelligent & Robotic Systems, 79(3-4), p.507.
- [33] S.K. Lee, K.R. Kim and J.H. Yu, (2014). BIM and ontology-based approach for building cost estimation. Automation in construction, 41, pp.96-105.
- [34] W. Mazairac and J. Beetz, (2013). BIMQL-An open query language for building information models. Advanced Engineering Informatics, 27(4), 444-456.
- [35] K. Patroumpas, G. Giannopoulos, S. Athanasiou, (2014). Towards GeoSpatial semantic data management: strengths, weaknesses, and challenges ahead. SIGSPATIAL/GIS 2014. 301-310.
- [36] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van de Walle and J. Van Campenhout, 2011. A semantic rule checking environment for building performance checking. Automation in Construction, 20(5), pp.506-518.
- [37] P. Pauwels, T. M. de Farias, C. Zhang, A. Roxin, J. Beetz, J. De Roo and C. Nicolle, (2017). A performance benchmark over semantic rule checking approaches in construction industry. Advanced Engineering Informatics, Volume 33, 2017, Pages 68-88.
- [38] P. Pauwels and A. Roxin, (2016). SimpleBIM: From full ifcOWL graphs to simplified building graphs. eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2016: Proceedings of the 11th European Conference on Product and Process Modelling (ECPPM 2016), Limassol, Cyprus, 7-9 September 2016.
- [39] P. Pauwels and W. Terkaj, (2016). EXPRESS to OWL for construction industry: towards a recommendable and usable ifcOWL ontology. Automation in Construction, 63, 100-133.
- [40] P. Pauwels, S. Zhang, Y.C. Lee, (2017). Semantic web technologies in AEC industry: A literature overview, Automation in Construction, Volume 73, January 2017, Pages 145-165.
- [41] M. Perry and J. Herring, (2012). OGC GeoSPARQL-A geographic query language for RDF data. OGC Implementation Standard. Sept.
- [42] M. J. Pratt, (2001). Introduction to ISO 10303-21 the STEP standard for product data exchange. Journal of Computing and Information Science in Engineering, 1(1), pp.102-103.
- [43] J. De Roo, (2011). Euler Yet another proof Engine. Available online: <http://eulerssharp.sourceforge.net/>. Last accessed on 12 June 2015.
- [44] B. Regalia, K. Janowicz and S. Gao, (2016). VOLT: A Provenance-Producing, Transparent SPARQL Proxy for the On-Demand Computation of Linked Data and its Application to Spatio-temporally Dependent Data. ESWC 2016: 523-538.
- [45] D. Rillaer van, J. Burger, R. Ploegmakers and V. Mitossi, (2012). Rgd BIM Standard, version 1.0.1. 1-29.
- [46] H. Schevers and R. Drogemuller, (2005). Converting the Industry Foundation Classes to the Web Ontology Language, in: Semantics, Knowledge and Grid, 2005. SKG'05. First International Conference on. Beijing, China Nov. 27, 2005 to Nov. 29, 2005
- [47] W. Shen, Q. Hao, H. Mak, J. Neelamkavil, H. Xie, J. Dickinson, R. Thomas, A. Pardasani, H. Xue, (2010). Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review, Advanced Engineering Informatics, Volume 24, Issue 2, 2010, pp. 196-207.
- [48] W. Solihin, C. Eastman, Y.C. Lee, (2015). Toward robust and quantifiable automated IFC quality validation. Advanced Engineering Informatics, Volume 29, Issue 3, pp. 739-756.
- [49] SPIN. (2011). SPARQL Inferencing Notation. <http://spinrdf.org/>.
- [50] Statsbygg, (2011). Statsbygg Building Information Modelling Manual Version 1.2. Available at: <http://www.statsbygg.no/bim>, accessed January 2014.
- [51] Solibri, (2000) . Solibri Model Checker . Available at: <https://www.solibri.com/products/solibri-model-checker/>, accessed January 2016.
- [52] M. Venugopal, C.M. Eastman, R. Sacks, J. Teizer, (2012). Semantics of model views for information exchanges using the industry foundation class schema, Advanced Engineering Informatics, Volume 26, Issue 2, April 2012, Pages 411-428.
- [53] R. Verborgh, M. V. Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert, (2016). Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. Web Semantics: Science, Services and Agents on the World Wide Web. 37-38: 184-206 (2016).
- [54] W3C (2014). SPARQL Extensions Computed Properties. https://www.w3.org/wiki/SPARQL/Extensions/Computed_Properties.
- [55] W3C (2015). Generating RDF from Tabular Data on the Web. <https://www.w3.org/TR/2015/WD-csv2rdf-20150416>.
- [56] A. Yurchyshyna and A. Zarli, (2009). An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction. Automation in Construction, 18(8), pp.1084-1098.
- [57] B.T. Zhong, L.Y. Ding, H.B. Luo, Y. Zhou, Y.Z. Hu, H.M. Hu, (2012). Ontology-based semantic modeling of regulation constraint for automated construction quality compliance checking, Automation in Construction, Volume 28, Pages 58-70.
- [58] C. Zhang, J. Beetz, M. Weise (2015). Interoperable validation for IFC building models using open standards, ITcon Vol. 20, Special issue ECPPM 2014 - 10th European Conference on Product and Process Modelling, pg. 24-39.

- [59] C. Zhang, and T. Chen (2001). Efficient feature extraction for 2D/3D objects in mesh representation. In Image Processing, 2001. Proceedings. 2001 International Conference on (Vol. 3, pp. 935-938). IEEE.
- [60] S. Zhang, F. Boukamp and J. Teizer, (2015). Ontology-based semantic modeling of construction safety knowledge: Towards automated safety planning for job hazard analysis (JHA). Automation in Construction, 52, pp.29-41.

Appendix

A. Resources

The vocabularies, rules and models are published with DOI: <https://doi.org/10.17605/OSF.IO/V5ENM>. Related source code for the backend is published on: <https://github.com/BenzclyZhang/BimSPARQL>.

B. Compared SPARQL queries

```
SELECT ?storey (COUNT(?wall) AS ?q)
WHERE{
  ?window a ifc:IfcWindow .
  ?r ifc:relatedBuildingElement ?window .
  ?r ifc:relatingOpeningElement ?opening .
  ?r a ifc:IfcRelFillsElement .
  ?rel ifc:relatedOpeningElement ?opening .
  ?rel ifc:relatingBuildingElement ?wall.
  ?rel a ifc:IfcRelVoidsElement .
  ?wall a ifc:IfcWall .
  FILTER NOT EXISTS {
    ?r1 ifc:relatedElements ?window .
    ?r1 ifc:relatingStructure ?storey.
    ?r1 a ifc:
      IfcRelContainedInSpatialStructure .
    ?r2 ifc:relatedElements ?wall .
    ?r2 ifc:relatingStructure ?storey.
    ?r2 a ifc:
      IfcRelContainedInSpatialStructure .
    ?storey a ifc:IfcBuildingStorey .
  }
}
```

Listing 12: Query to count load bearing walls for each storey

```
SELECT ?storey (COUNT(?wall) AS ?q)
WHERE{
  ?storey a ifc:IfcBuildingStorey .
  ?rel ifc:relatingStructure ?storey .
  ?rel a ifc:
    IfcRelContainedInSpatialStructure .
```

```
?rel ifc:relatedElements ?wall .
?wall a ifc:IfcWall .
{
  ?r ifc:relatedObjects ?wall.
  ?r a ifc:IfcRelDefinesByProperties .
  ?r ifc:relatingPropertyDefinition ?pset .
  ?pset a ifc:IfcPropertySet .
  ?pset ifc:name ?n .
  ?n expr:hasString "Pset_WallCommon" .
  ?pset ifc:hasProperties ?property .
  ?property a ifc:IfcPropertySingleValue .
  ?property ifc:name ?name .
  ?name expr:hasString "LoadBearing" .
  ?property ifc:nominalValue ?value .
  ?value expr:hasBoolean true .
}UNION{
  ?r ifc:relatedObjects ?wall.
  ?r a ifc:IfcRelDefinesByType .
  ?r ifc:relatingType ?type .
  ?type ifc:hasPropertySets ?pset .
  ?pset a ifc:IfcPropertySet .
  ?pset ifc:name ?n .
  ?n expr:hasString "Pset_WallCommon" .
  ?pset ifc:hasProperties ?property .
  ?property a ifc:IfcPropertySingleValue .
  ?property ifc:name ?name .
  ?name expr:hasString "LoadBearing" .
  ?property ifc:nominalValue ?value .
  ?value expr:hasBoolean true .
  FILTER NOT EXISTS{
    ?r2 ifc:relatedObjects ?wall.
    ?r2 a ifc:IfcRelDefinesByProperties .
    ?r2 ifc:relatingPropertyDefinition ?pset2 .
    ?pset2 a ifc:IfcPropertySet .
    ?pset2 ifc:name ?n2 .
    ?n2 expr:hasString "Pset_WallCommon" .
    ?pset2 ifc:hasProperties ?property2 .
    ?property2 a ifc:IfcPropertySingleValue .
    ?property2 ifc:name ?name2 .
    ?name2 expr:hasString "LoadBearing" .
  }
}GROUP BY ?storey
```

Listing 13: Query to count load bearing walls for each storey

C. SPIN rules for implementing functions for case in section 7

```
SELECT (?a/?wallArea AS ?Ap)
WHERE{
  ?arg1 pdt:hasGrossWallArea ?wallArea .
  { SELECT (SUM(?windowArea) AS ?area){
    ?window schm:isPlacedIn ?arg1 .
    ?window pset:fireRating "OH-45" .
    ?window pdt:hasWindowArea ?windowArea .
```



```
} GROUP BY ?arg1 }
}
```

Listing 14: Query to implement the function *ibc:hasAp* referenced in Listing 11

```
SELECT (?a/?wallArea AS ?Au)
WHERE{
  ?arg1 pdt:hasGrossWallArea ?wallArea .
  { SELECT (SUM(?windowArea) AS ?area){
    ?window schm:isPlacedIn ?arg1 .
    FILTER NOT EXISTS{
      ?window pset:fireRating "OH-45" .
    }
    ?window pdt:hasWindowArea ?windowArea .
  } GROUP BY ?arg1 }
}
```

Listing 15: Query to implement the function *ibc:hasAu* referenced in Listing 11

```
SELECT (MIN(?d) AS ?distance)
WHERE{
  ?line a ifc:IfcAnnotation .
  ?line ifc:name ?name .
```

```
?name expr:hasString "Lot Line" .
(?arg1 ?line) spt:distanceXY ?d .
}GROUP By ?arg1
```

Listing 16: Query to implement the function *ibc:hasFireSeparationDistance* referenced in Listing 11

```
SELECT ?ap
WHERE {
  ?b1 ibc:minFSDistance ?min .
  ?b1 ibc:maxFSDistance ?max .
  ?b1 ibc:openingProtection ?arg2 .
  ?b1 ibc:sprinklerProtection ?arg3 .
  ?b1 ibc:allowableArea ?ap .
  FILTER ((qudtspin:convert(?arg1, unit:Meter,
    unit:Foot) >= xsd:double(?min)) && (
    qudtspin:convert(?arg1, unit:Meter, unit:
    Foot) < xsd:double(?max))) .
}
```

Listing 17: Query to implement the function *ibc:allowableArea_T705-8* referenced in Listing 11