Managing Space Requirements of New Buildings Using Linked Building Data Technologies

Rasmussen, Mads Holten; Hviid, Christian Anker; Karlshøj, Jan; Bonduel, M.

Publication date:
2018

Document Version
Peer reviewed version

Managing Space Requirements of New Buildings Using Linked Building Data Technologies

M.H. Rasmussen, C.A. Hviid & J. Karlshøj
Department of Civil Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark

M. Bonduel
Department of Civil Engineering, Technology Cluster Construction, KU Leuven, Ghent, Belgium

ABSTRACT: Any stakeholder operating in the AEC industry knows that designing a building is a complex and highly iterative task. The project evolves over time and changes happen rapidly, meaning that design requirements, as well as solutions (often as a consequence), must undergo revision. Since building requirements are, however, documented and handled in a predominantly manual manner, the work processes are not aligned with the dynamic nature of the projects. Tracking and acting upon changes is a manual, and therefore an error-prone and labour intensive task. In this article, we suggest a generic method for working with the concept of spaces at different abstraction levels in order to compare requirements with actual properties in a non-static manner using semantic web technologies, primarily developed by the W3C Linked Building Data (LBD) Community Group. The generic modelling approach has the potential of also being applied to other concepts than building spaces.

1 INTRODUCTION

When buying a product you can rightly expect it to correspond to the technical specifications on which the purchase was originally based. When buying a building, however, the reality is unfortunately not always so (Kiviniemi 2005). Bertelsen (2003) describes construction as a complex system because of three main characteristics: (1) autonomous agents (2) undefined values and (3) non-linearity. Delivering a complete product specification in the form of a building program at day 1 is nearly impossible as everyone gains knowledge and insights as the design evolves, and as a result, the building program itself cannot be static during the design. The documentation and handling of it, therefore, needs to be dynamic, which is unfortunately typically not the case (Kiviniemi 2005). The majority of building design processes are today characterized by manual information extraction from static documents, and as the design progresses it becomes a cumbersome task for the project participants to keep track of, and meet the evolving client requirements. Because of the predominantly manual information handling, the quality of information exchange between project stakeholders is furthermore highly determined by the social capabilities and communicative skills of the individual practitioners (Bendixen 2007). This is a challenge that the methodology of Building Information Modelling (BIM) will hopefully remedy over time. However, unfortunately the BIM authoring tools of today are not delivering satisfactory interoperability, and data is therefore often trapped in data silos (Terkaj 2017).

In this article, we first provide a brief overview of existing software and data modelling approaches that focus on building requirements specification. We then argue why we believe semantic web technologies can possibly provide the means to overcome current challenges when dealing with the dynamic behaviour of building requirements. Based on knowledge manually deduced from existing document-based building programs and discussions with practitioners in the consulting engineering company, Niras, we have defined a set of competency questions. These were used as constraints for what the data model should be capable of. The model was developed accordingly, chiefly by using terminology defined in already existing and widely adopted ontologies. Lastly, we developed a set of tests to evaluate the modelling approach on the Common BIM Model "Duplex Apartment"[1]. The dataset was established partly by manually defining requirements as

[1] https://www.nibs.org/?page=bsa_commonbimfiles#project1

an RDF-graph (Resource Description Framework) following the suggested modelling approach, and partly by using a custom developed exporter for the BIM authoring tool, Revit[2]. The latter establishes an RDF-graph using ontologies provided by the World Wide Web Consortium Linked Building Data Community Group (W3C LBD-CG).

1.1 *Open standards*

The effort of storing knowledge in a construction project, including the information exchange between its stakeholders, has been addressed by the buildingSMART organisation. With standards such as Industry Foundation Classes (IFC) (Liebich and Wix 1999), Information Delivery Manuals (IDM) and Model View Definitions (MVD) they deliver a solid framework for information exchange and storage.

The W3C also has made efforts to standardize information exchange using semantic web technologies such as the Web Ontology Language (OWL) to construct formal vocabularies to describe a certain domain of interest. The scope of these technologies is not limited to the AEC industry alone, and therefore researchers and practitioners from a wide variety of domains are contributing to their continuous development.

One main difference between the above two methodologies is that OWL relies on an Open World Assumption (OWA), meaning that the schema can evolve over time to include concepts not initially thought of. This is quite different from typical database systems that depend on a Closed World Assumption (CWA) for defining schemas, such as IFC. Another benefit is that the full dataset does not need to be available at one location but can be combined with other datasets as needed, being both Linked Open Datasets (LOD) available online (material data, weather data, geographical data etc.) and private datasets, possibly hosted by other project stakeholders. Owners of such private datasets can restrict the access to specific partners.

The W3C Resource Description Framework (RDF) standard is used to describe Linked Data in a directed graph consisting of a collection of triples. A triple has three parts: a node (the subject), an edge (the predicate) and another node (the object) connected to the first node through the predicate-edge. All sub-elements of a triple are made globally unique by denoting them with a Uniform Resource Identifier (URI) except for objects that are literal values such as strings, integers, Booleans etc. The datatype of such literals are also described with a URI, and is often defined in an ontology version of the Extensible Markup Language (XML) Schema Definition (XSD). Both the terminology layer (TBox) - including semantics for classes and properties, and the data layer (ABox), covering individual instances and their interrelations, are described using RDF. The W3C encourages developers to make their ontologies publicly available so that useful ontology-related information can be retrieved from the URI. To continue, the W3C recommends that terms from widely adopted ontologies are used to explicitly describe the data layer.

An RDF graph is traversed using the SPARQL Protocol and RDF Query Language (SPARQL) and if it is described using widely adopted ontologies it is possible to structure generic, globally applicable queries to deduce knowledge. The semantics described in the TBox also allow reasoning engines to deduce implicit knowledge from what is explicitly defined in the ABox. A simple example: If `chair` is a sub-class of `furniture` (TBox), then all instances of `chair` are also instances of `furniture` (ABox).

1.2 *Cloud-based BIM solutions*

Although building programs are typically defined in static documents (Word, PDF) there are a few cloud-based BIM applications for building requirements management on the market. They typically consist of a user interface (UI) that enables the user to do create, read, update and delete (CRUD) operations on requirements stored in a central database along with a communication link to native BIM authoring tools. Since each internal database has a closed proprietary schema rather than a schema defined according to the previously described open standards, interlinking the requirements to information that exists outside the application is not easily accomplished. Additionally, migrating from one tool to another is seen as a cumbersome task. Some applications do offer a REST (representational state transfer) API (application programming interface) providing a machine-accessible interface to the internal data model. However, the design of this interface is also following a proprietary schema and therefore a deep understanding of this schema is a prerequisite for interpreting and using the data in other applications.

Onuma and dRofus are examples of BIM applications for requirements management that offer a

[2] https://github.com/MadsHolten/revit-bot-exporter

REST API to interact with the data model[3,4], and they use XML and JavaScript Object Notation (JSON) respectively as data format. Both APIs offer only limited interaction with the data model and although accessible from outside, they are tightly coupled to their native data models.

The SPARQL Protocol (Feigenbaum et al. 2013) and SPARQL Graph Store HTTP protocol (Chimezie Ogbuji 2013) are W3C recommendations specifying how to make an RDF-graph available through a REST architecture. Accessing the graph is achieved by sending a SPARQL query to a URI hosting a SPARQL endpoint, and this provides an interface for clients to do CRUD operations on the dataset. A cloud-based BIM tool using the W3C open standards to describe the schema could host a SPARQL endpoint in order to allow clients to access the data model using standardised SPARQL queries, but to our knowledge, no such tool currently exists.

1.3 *Linked Building Data*

Research has provided us with several examples of how semantic web technologies can be used to enhance data handling in the AEC industry. The typical research contribution is an ontology which describes a subset of the construction domain with a distinct scope such as smart homes and sensor data or even the construction domain as a whole. Pauwels & Terkaj (2016) proposed ifcOWL as the OWL-based counterpart for the IFC schema and probably the most widely adopted ontology in the AEC domain.

It has later been argued that this quite literal conversion of the IFC schema is not appropriate as it (1) contains artefacts from the EXPRESS schema from which it originates making queries less logic and (2) describes too wide a scope, thereby violating the W3C best practice of omitting redundancy and making it hard to get familiarized with (Pauwels & Roxin 2016; Rasmussen et al. 2017a).

Another, more modular approach for building-related ontologies is suggested by the W3C LBD-CG. A minimal ontology, the Building Topology Ontology (BOT) (Rasmussen et al. 2017a) describes the main concepts of a building and thereby serves as an extensible core for describing any concept in its context of a building. Another ontology, PROPS, describes building-related properties and is at the time of writing a conversion of the properties contained in the IFC4 schema[5]. The conversion approach is also used in the PRODUCT ontology which describes building-related products. Finally, the Ontology for Property Management (OPM) extends concepts from the Smart Energy-Aware Systems (SEAS) ontology to provide the means to describe property reliability as well as property changes over time using property states.

Both the IFCtoLBD-converter[6] (Bonduel et al. 2018) and an exporter for Revit[7] (Rasmussen et al. 2017b) generate LBD compliant RDF triples from conventional BIM models.

In this study we have used and extended a set of widely adopted web ontologies for property handling (schema.org/goodrelations), provenance data (PROV-O), literal units (Unified Code for Units of Measure (UCUM) (Lefrançois 2018)) along with the earlier mentioned LBD ontologies. Using these ontologies in combination with OWL description logics, we illustrate an approach for specifying project specific space classes that explicitly state the client's requirements. We further show how the architectural spaces can automatically inherit requirements based on the class they are assigned to using standard OWL reasoning engines. Queries to compare and evaluate requirements to actual properties of the space instances are further illustrated and a simple use case, is presented to simulate both requirement and property changes and the handling of these.

2 REQUIREMENTS MODELLING

In this section we illustrate how concepts defined in the BOT, OPM and schema.org ontologies can be used to model space requirements. Initially, various client requirements specifications for construction projects in which Danish consulting company Niras has been involved, were reviewed. In these specifications, it is common practice to specify space requirements at *type* level rather than at *instance* level.

IFC and various BIM authoring tools use the concept of *types* and include a mechanism for inheriting properties of a *type* to *instances* belonging to that *type*. *Instances* can further extend the set of properties at an individual level and properties can even be overridden (Borgo et al. 2014). It is clear that the instances belong at ABox level, but the concepts of

[3] http://www.onuma-bim.com/platform/api
[4] https://wiki.drofus.com/display/DV/REST+API
[5] https://github.com/w3c-lbd-cg/props/blob/master/IFC4-output.ttl
[6] https://github.com/jyrkioraskari/IFCtoLBD
[7] https://github.com/MadsHolten/revit-bot-exporter

space and object *types* are less obvious. In BIM tools, space and object *type instances* are defined at the data layer rather than the schema layer, but from an ontology engineering perspective, it would arguably be more correct to consider the *type instances* themselves at schema level.

In the following section, we will investigate a TBox modelling approach of *space types* that must be capable of providing answers to the following competency questions:

- *CQ1:* How to model a *space type*?
- *CQ2:* How to assign a quantitative requirement to a *space type*?
- *CQ3:* How to state that a *designed space* instance matches a *space type* of the client's requirements specification?
- *CQ4:* How to check if a property that also exists as a requirement is fulfilled by the architectural design?
- *CQ5:* How to check an adjacency or quantity requirement?
- *CQ6:* How to update a *space type* and its assigned requirements?

2.1 *CQ1: Modelling a space type*

Modelling a *space type* is achieved by defining a project-specific extension of BOT, in this case in the namespace of the building client. In Figure 1 the class `client:spacetype_bathroom1` is defined as a sub-class of `bot:Space` meaning that any instance of the class will be classified as a `bot:Space`. The `rdfs:label` and `rdfs:comment` are widely adopted predicates from the RDF Schema (RDFS) that provide a human-readable specification of the class. In this example, in Danish and English language.

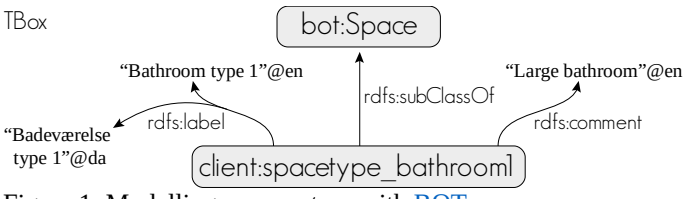

Figure 1. Modelling a *space type* with BOT.

2.2 *CQ2: Assigning a quantitative requirement*

In order to meet the demands for modelling a *space requirement*, it should be possible to capture the following information:

- Range, (minimum and maximum) or specific value to be matched
- Quantitative unit of the value
- Property changes over time (deleted, modified)

OWL includes logics to describe property restrictions for classes. For example, it is possible to describe that `:BlueCars` is a sub-class of all cars that have a blue color, which entails that every instance of the `:BlueCars` class will consequently be blue. Figure 2 illustrates how an `owl:Restriction` can be used to describe that all instances of `client:spacetype_bathroom1 have a props:area` with the value `client:property_001`. This objectified property belongs to the ABox of the client's dataset, which allows it to evolve over time.

Rasmussen et al. (2018) describe three levels of complexity for assigning properties to some feature of interest (FoI). Level 3, the most expressive form, satisfies the demand of allowing property changes over time and is therefore used to model *space requirements*. Figure 2 illustrates how the property has a property state (`client:state_p001_001`) assigned. This state is currently classified as the `opm:CurrentPropertyState`, which indicates that it is the most recent state of the property but this might change over time as the client requirements are revised. A new class `opm:Required` which we suggest to implement as an extension of OPM is used to specify that the state is a requirement rather than a designed property. A value range is specified using properties defined in schema.org and the generation time is captured using PROV-O. The unit is given as part of the value string using a custom datatype based on UCUM. Further metadata such as who created the property state for which reason can also be attached.

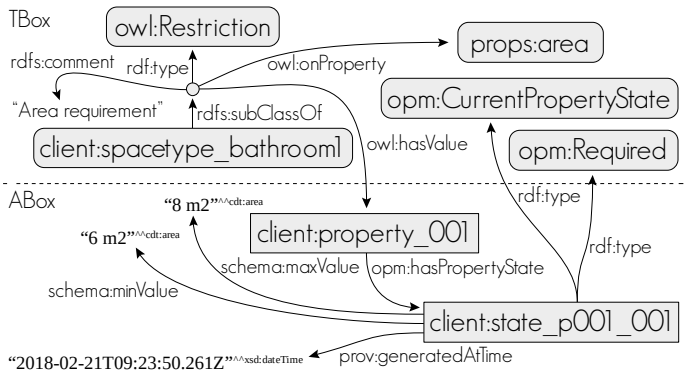

Figure 2. Assigning a requirement using (Rasmussen et al. 2018) Level 3.

2.3 *CQ3: Mapping designed space instances to spaces requested by the client*

At one point, as the architectural design progresses, the architect's dataset will hold a number of *designed spaces* that should match the *space types* required by the client. At this point, the architectural spaces are geometrically defined, and therefore they have an actual area.

Mapping a *designed space* to a client *space type* is handled by stating that the *designed space* is an instance of the specific *space type* class. Figure 3 illustrates how properties of the client *space type* (`client:spacetype_bathroom1`) are inherited to all instances of this class. In this example, spaces `inst:room123` and `inst:room213` both inherit `client:property_001` (and its property state) as the value for property `props:area`.

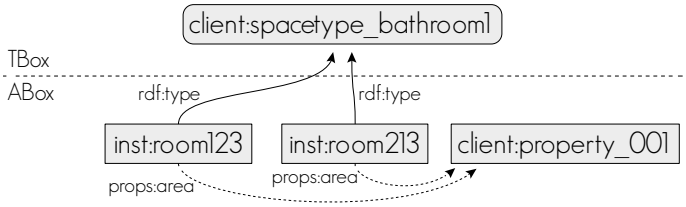

Figure 3. Two designed spaces are classified as client: spacetype_bathroom1. Therefore the properties (requirements) of the client *space type* are inherited by the *designed spaces*.

2.4 *CQ4: Checking that a requirement is fulfilled*

When the same space property exists both as a requirement and a designed property it is possible to do a comparison in order to check if the requirement is met. Figure 4 illustrates `inst:room123` which has the property `props:area` assigned twice. Explicitly as a result of its geometry and implicitly as a requirement inherited by the mechanism described in Figure 3. Performing the comparison is possible by traversing the graph using a SPARQL query.

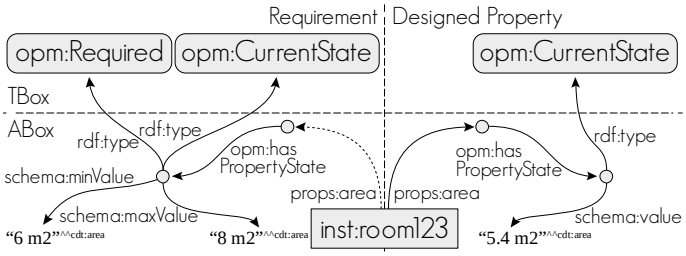

Figure 4. Requirement vs. property.

Listing 1 shows a SPARQL query to retrieve all violations of the `props:area` requirement in the model, when both the requirements and designed properties are all in one database. The query is structured as a graph traversal which operates by matching the defined patterns. The first triple pattern maps anything that is an instance of `bot:Space` to the variable `?space`. The next pattern is a sub-query which is used to get data from the state of `props:area` that is classified as `opm:Required`. The variable `?space` is used to match the same space, and the URI of the property object is mapped to variable `?reqURI`. All states of the property are assigned to variable `?reqState` but the next two triples limit the result to only include the one state which is both classified as `opm:Required` and `opm:CurrentPropertyState`. Since a requirement can be specified either as an exact match or as a range, each of the `schema:value` patterns are optional.

A similar pattern is used to get the actual property and by using a filter it is ensured that the requirement is not assigned to variable `?propURI` (since both match the pattern). The value of `?propURI`'s latest state is assigned to variable `?val` and compared to the required range to check if it is violated. A result is returned only if the requirement is violated.

Replacing `?space` with `inst:room123` or the URI of any other space will return violated requirements for this particular space and this approach can be used to switch any variable with a constant.

Listing 1. SPARQL query to retrieve violated requirements

```
SELECT *
WHERE {
  # Must be a space
  ?space rdf:type bot:Space .

  # Sub-query to get requirement
  {
    SELECT ?space ?reqURI ?reqVal ?reqMax ?reqMin
    WHERE {
      ?space    props:area           ?reqURI .
      ?reqURI   opm:hasPropertyState ?reqState .
      ?reqState rdf:type opm:Required .
      ?reqState rdf:type opm:CurrentPropertyState .
      OPTIONAL {?reqState schema:value ?reqVal}
      OPTIONAL {?reqState schema:minValue ?reqMin}
      OPTIONAL {?reqState schema:maxValue ?reqMax}
    }
  }

  # Get property
  ?space props:area ?propURI .
  FILTER(?propURI != ?reqURI) # Disjoint from req
  ?propURI opm:hasPropertyState ?propState .
  ?propState rdf:type opm:CurrentPropertyState .
  ?propState schema:value ?val

  # Compare requirements to actual value
  BIND( ?value !=  ?reqVal AS ?matchViolated   )
  BIND( ?value <?reqMin    AS ?minViolated     )
  BIND( ?value >?reqMax    AS ?maxViolated     )

  # Show only results where a requirement is
  # violated
  FILTER( ?matchViolated || ?minViolated ||
                                  ?maxViolated )
```

2.5 *CQ5: Adjacency and quantity requirements*

Specifying adjacency or quantity requirements is not different from any other requirement. However, special queries must be used to check whether these are violated. The same is the case for other requirements such as zone or element containment.

Checking if the required quantity of spaces of a certain *space type* is met, is accomplished by the query shown in Listing 2. Accessing the requirement can

be done in the main query since it is not necessary to distinguish between two properties of the same kind, but in order to count the number of *space type* occurences, a sub-query is necessary. Listing 2 shows the optional sub-query to count the number of *designed space* instances per client *space type*. Each *space type* is assigned a unique `?reqURI` for the `props:quantity` property requirement, so this can be used for the grouping. This query is executed before continuing to the next step where requirement `props:quantity` is compared to `?value`.

Listing 2. Sub-query to count number of *designed space* instances that have a specific quantity requirement assigned.

```
{
  SELECT ?reqURI (COUNT(?reqURI) AS ?qty)
  WHERE {
    ?space props:quantity ?reqURI .
  } GROUP BY ?reqURI
}
```

Finding violated adjacency requirements is likewise handled by first getting the requirement (like illustrated in Listing 1). Also in this case it can be done in the main query, and this time it is only necessary to get the `schema:value` and bind it to `?reqVal`. By using the MINUS clause a result is only returned if the space does not have an adjacency to a *designed space* defined as an instance of the required client *space type*.

Listing 3. SPARQL query to retrieve violated adjacency requirements.

```
# Return result if the space does not have an
# adjacent space of the required type
MINUS {
  ?space    bot:adjacentZone ?adjSpace .
  ?adjSpace rdf:type         ?reqVal .
}
```

2.6 *CQ6: Performing updates*

Changes to property requirements are according to Rasmussen et al. (2018) handled by creating a new current state and removing the `opm:CurrentPropertyState` from the evaluation that was previously defined as the current state. This can be achieved with an update query which can be generated using the OPM query generator JavaScript library[8]. Since all the queries explicitly look for the current state of both properties and requirements, the evaluations will automatically reflect the changes.

[8] https://www.npmjs.com/package/opm-qg

3 USE CASE

To illustrate a possible workflow for modelling, mapping and evaluating requirements, a simple use case was set up. The Common BIM Model "Duplex Apartment" was used as a reference, and the Revit BOT exporter[9] plugin (Rasmussen et al. 2017b) was extended to include the concept of *space types* and OPM *property states*. The exporter was used to export the architectural model in LBD format. The steps to establish the dataset were the following:

1) Define client requirements in RDF. (This step should preferably be accomplished through a UI)
2) Run BOT exporter in Revit to:
   - Create and assign a Revit URI parameter to spaces and elements
   - Create Revit SpaceTypeURI parameter
   - Export BOT relationships and properties to RDF
3) Specify *space type* URI corresponding to the URI used for the client *space type* and re-export triples (Figure 5)
4) Use Dynamo script to export zone adjacencies to RDF. This functionality will be implemented in the exporter plugin in the future

Figure 5. Revit shared parameters for URI and SpaceTypeURI.

Once the dataset was available it was loaded into a triplestore in order to do the checks described in the previous section. The checking is implemented in a JavaScript based testing tool that is available online[10], while the results are presented here.

3.1 *Testing property requirements*

All *space types* in the test have an area requirement specified. In general, the areas are fulfilled by the designed spaces, except for `inst:spacetype_bedroom` and `inst:spacetype_bathroom1`. Once the dataset is loaded into the triplestore, the tool performs the query from Listing 1 to find area requirement violations. Listing 4 shows the results.

Listing 4. Test tool output for violated property requirements. Numbers in parenthesis are (actual/range).

```
- 'Bathroom 2 B204' violates req. (5.44/(6-))
- 'Bathroom 2 A204' violates req. (5.42/(6-))
```

[9] https://github.com/MadsHolten/revit-bot-exporter
[10] www.student.dtu.dk/~mhoras/ecppm2018/test.zip

```
- 'Bedroom 1 B202'  violates req. (26.12/(20-25))
- 'Bedroom 2 B203'  violates req. (26.18/(20-25))
- 'Bedroom 2 A203'  violates req. (26.18/(20-25))
- 'Bedroom 1 A202'  violates req. (26.12/(20-25))
```

3.2 *Checking quantity of spaces*

Some *space types* have a requirement for quantity of *designed space instances*, and for `inst:spacetype_living_room` a requirement of seven occurrences is specified, which is not fulfilled in the case of the Duplex house model. A query to group the rooms into apartments based on the room numbers (which are suffixed with either A or B) was implemented in the test tool. The query from Listing 2 was modified slightly in order to accommodate this before counting the number of *designed space* occurrences for each *space type*. The result of this query was, correctly, that the requirement was not met as there is only one living room per apartment in the Duplex model.

Listing 5. Test tool output for violated quantity requirements. Numbers in parentheses are (actual/range).

```
- 'Living Room A102' (1/7)
- 'Living Room B102' (1/7)
```

3.3 *Testing adjacency requirements*

Two adjacency requirements were given as a client requirement:

– `spacetype_living_room`/`spacetype_kitchen`
– `spacetype_bedroom`/`spacetype_bathroom1`

The query from Listing 3 revealed that requirement 2 is only fulfilled by one of the bedrooms in each apartment, which is correct.

3.4 *Changing requirements*

By performing four SPARQL update queries, three client requirements were revised and a new one was added:

– Area requirement for `spacetype_bathroom1` relaxed from 6 m^2 to 5 m^2.
– `props:quantity` for `spacetype_living_room` relaxed from 7 to 1.
– New *space type* `spacetype_bedroom2` with `props:quantity` requirement of 1 and area requirement of minimum 9 m^2 added.
– Adjacency requirement between `spacetype_bedroom` and `spacetype_bathroom1` deleted by appending new `opm:PropertyState` of class `opm:Deleted`.

Re-running the tests from section 3.1 and 3.2 now concludes that the area requirements of `'Bathroom 2 A204'` and `'Bathroom 2 B204'`, the `props:quantity` requirement for `spacetype_living_room` and the adjacency requirements for `'Bedroom 1 A202'` and `'Bedroom 1 B202'` are no longer violated.

The requirements for the new bedroom type cannot be evaluated with the queries presented in Section 2 since the class is not assigned to any spaces. In order to check for required spaces which have not been instantiated, one must do a query starting from the client *space type* itself, and even though this is less intuitive, it is possible. Listing 6 shows a query pattern to retrieve a *space type* which has a quantity requirement assigned, but is not instantiated.

Listing 6. Find *space types* with a quantity requirement but no instances.

```
# GET QUANTITY REQUIREMENT
?spaceType rdfs:subClassOf [
  rdf:type owl:Restriction ;
  owl:onProperty props:quantity ;
  owl:hasValue ?reqURI
] .
MINUS { ?space a ?spaceType }
```

Since the initial requirements are all available in the model, the architect is able to track the changes and relate a property compliance check to a certain state of a requirement.

4 CONCLUSIONS AND FUTURE WORK

The main outcome of this work is the illustration of how to use semantic web technologies and existing ontologies, BOT in particular, to establish a knowledge model of requirements for spaces of a new building. The model illustrates an approach to describe space requirements at *type level* in a way that utilizes OWL reasoning capabilities thereby providing best practice examples of how to extend BOT at project level.

In the use case presented in this work, *designed architectural spaces* inherit properties of the *space types* described by the client. The same approach could be used for (1) other features of interest such as building elements or the building as a whole or (2) other generalisations such as an automation control strategy. In the use case, the requirements were modelled manually, but it is obviously not practical for practitioners to do this, so some CRUD application with a user-friendly UI should be developed.

Another interesting use case to investigate is *derived requirements*. Specific requirements such as minimum and maximum temperature, fresh air supply etc. are a result of the more general requirement; the desired indoor climate class (according to EN15251) and can be deduced by taking into account properties of the users of the space (ie. activity level, clothing). The specific indoor climate requirements set the constraints for the technical systems to be designed by the HVAC engineer, and modelling these interdependencies could potentially provide a valuable tool for *design change consequence analysis*.

In the use case, all data was stored in the same triplestore, but in a real world implementation the client would probably make the project specific classes and associated requirements available to project participants as a SPARQL-endpoint hosted on a separate server or as part of a Common Data Environment (CDE). Further research in how such an implementation could be configured is a separate research topic.

The use of OPM enables documentation of design and requirement changes over time, and in the use case it was used to revise requirements. Inferring into the graph that a requirements check was made based on a specific state of a requirement could be used for documentation purposes, but this was out of the scope for this work. The legal aspects of being able to document design changes, potentially in combination with block chain technology could entail great benefits and composes a separate research topic.

In summary, this work illustrates a data modelling approach that provides all the means to overcome current challenges when dealing with evolving design data and requirements in the complex construction industry. It is our belief that future BIM tools can benefit from adopting these technologies and methodologies.

5 ACKNOWLEDGEMENTS

Special thanks to the NIRAS ALECTIA Foundation and Innovation Fund Denmark for funding.

6 REFERENCES

Bendixen, M. 2007 The challenges of consulting engineers. *PhD Thesis.* Kgs. Lyngby: Technical University of Denmark.

Bertelsen, S. 2003 Construction as a Complex System. *Proceedings of IGLC* 11(February):143–68.

Bonduel, M., Oraskari, J. & Pauwels, P. 2018 The IFC to Linked Building Data Converter - Current Status. *6th Linked Data in Architecture and Construction Workshop (LDAC).*

Borgo, S. et al. 2014 Towards an Ontological Grounding of IFC *6th Workshop Formal Ontologies Meet Industry, Joint Ontology Workshops, CEUR.*

Chimezie O. 2013 SPARQL 1.1 Graph Store HTTP Protocol. Retrieved May 15, 2018 (https://www.w3.org/TR/sparql11-http-rdf-update/).

Feigenbaum, L. et al. 2013 SPARQL 1.1 Protocol. Retrieved May 15, 2018 (https://www.w3.org/TR/sparql11-protocol/).

Kiviniemi, A. 2005 Requirements Management Interface to Building Product Models *PhD Thesis.* Stanford University.

Lefrançois, M. & Zimmermann, A. 2018 The unified code for units of measure in RDF: cdt:ucum and other UCUM datatypes *Proceedings of the International Semantic Web Conference (ISWC),* demonstration paper, submitted.

Liebich, T. & Wix, J. 1999 Highlights of the Development Process of Industry Foundation Classes. *Proceedings of the 1999 CIB W78 Conference*.

Pauwels, P. & Roxin, A. 2016 SimpleBIM : From Full IfcOWL Graphs to Simplified Building Graphs. *European Conference on Product and Process Modelling (ECPPM).*

Pauwels, P. & Terkaj, W. 2016. EXPRESS to OWL for Construction Industry: Towards a Recommendable and Usable IfcOWL Ontology. *Automation in Construction* 63:100–133. doi: 10.1016/j.autcon.2015.12.003.

Rasmussen, M. H. et al. 2017a Proposing a Central AEC Ontology That Allows for Domain Specific Extensions. *Lean and Computing in Construction Congress - Volume 1: Proceedings of the Joint Conference on Computing in Construction* 237–44. doi: 10.24928/JC3-2017/0153.

Rasmussen, M. H., et al. 2017b Web - Based Topology Queries on a BIM Model. *5th Linked Data in Architecture and Construction (LDAC2017) Workshop*. doi: 10.13140/RG.2.2.22298.95685.

Rasmussen, M. H. et al. 2018 OPM: An Ontology for Describing Properties That Evolve over Time. *6th Linked Data in Architecture and Construction Workshop (LDAC), CEUR.*

Terkaj, W. et al. 2017 Reusing Domain Ontologies in Linked Building Data : The Case of Building Automation and Control. *8th Workshop Formal Ontologies Meet Industry, Joint Ontology Workshops, CEUR*