Web-based topology queries on a BIM model

Rasmussen, Mads Holten; Hviid, Christian Anker; Karlshøj, Jan

Publication date:
2017

Document Version
Peer reviewed version

Web-based topology queries on a BIM model

Mads Holten Rasmussen[1], Christian Anker Hviid, and Jan Karlshøj

Technical University of Denmark, Copenhagen, Denmark

Abstract. Building Information Modeling (BIM) is in the industry often confused with 3D-modeling regardless that the potential of modeling information goes way beyond performing clash detections on geometrical objects occupying the same physical space. Lately, several research projects have tried to change that by extending BIM with information using linked data technologies. However, when showing information alone the strong communication benefits of 3D are neglected, and a practical way of connecting the two worlds is currently missing.
In this paper, we present a prototype of a visual query interface running in a web browser, that enables the user to gain a deeper understanding of what can be extracted from a Building Topology Ontology (BOT) knowledge base. The implementation enables the user to query the graph, and provides visual 3D-feedback along with simple table results.
The main purpose of the paper is to establish a baseline for discussion of the general design choices that have been considered, and the developed application further serves as a proof of concept for combining BIM model data with a knowledge graph and potentially other sources of Linked Open Data, in a simple web interface.

1 Introduction

Today's BIM software allows the user to (1) create geometrical objects (2) establish geometrical constraints and relationships between them and (3) assign properties to them. Some software also offers calculation and simulation capabilities making them a full-suite solution. The complexity of the BIM tools entail that it requires special training to use them, and the fact that the most experienced engineer typically lacks IT-competencies induces that in the industry, they are often operated by technical designers. As the tools further lack interoperability with the software commonly used by engineers, they are obstructed from utilising the full BIM potential. The result is that essential information gets trapped in proprietary BIM files or "digital cemeteries" as one might be tempted to call them, while engineers waste time and introduce human errors by doing manual information takeoffs.

Point (1) and (2), respectively creating geometrical objects and establishing geometrical constraints and relationships between them, are the key features of the BIM tools of today and they conquer this task very well. When it comes to (3) assigning properties to BIM objects, however, it is not completely clear why this task is best accomplished in a large BIM suite. If other tools could instead

subscribe to the geometrical properties in the model and get notified when these change as the design progresses, a potential efficiency gain could be achieved in typical working tasks. By making data accessible in the cloud, larger engineering companies and software vendors would be able to develop task-specific design tools that are linked directly to the one true source of information, thereby reading, manipulating and writing to a linked framework of data. The prototype documented in this paper serves as a proof of concept of such a tool.

1.1 BIM and the semantic web

The Building Topology Ontology (BOT)[1] suggested by the *World Wide Web Consortium (W3C) Linked Building Data Community Group* (W3C LBD-CG) is a compact ontology designed for expressing the semantics of a building [1]. It was a result of a study in existing ontologies in the construction, automation and smart cities domains, where it was discovered that basic semantics of buildings were redundantly described in all ontologies. Currently, there are no software implementations of the ontology, which makes it hard to communicate its potential to people who do not have a background in ontology design. To illustrate the features of the ontology and for BIM practitioners and engineers to imagine potential use cases, a simple web application and an exporter for the commercial BIM tool, Autodesk Revit[2], was developed. The exporter was developed to establish a BOT-compliant knowledge representation of a building from the software tool most widely used in danish consulting engineering companies, but the knowledge representation could also have been generated through the API of another BIM-tool, or by parsing a file in the open ISO standardised Industry Foundation Classes (IFC)[2] format. One could also imagine using the same approach as presented in this paper to align with other OWL representations of building data such as ifcOWL [3].

In the following sections we seek to document the design considerations and the overall system architecture of the application and in closing we cover our reflections on future work.

2 Building Topology

In the prototype, a subset of the semantics defined in BOT is extracted in order to describe relationships in the BIM model. Some relationships are directly accessible in the Revit API and some are not. While anything can be derived by performing operations on the geometry, it can be a time-consuming task both to develop and run the code and for the purpose of this case study, only the directly available relationships have been extracted.

[1] https://w3id.org/bot#
[2] www.autodesk.com/Revit

2.1 Class definitions

In the current implementation only some of the main elements are exported. Walls, doors and windows are exported as `bot:Elements`, levels as `bot:Storeys` and spaces/rooms as `bot:Spaces`. Revit Spaces are the Mechanical engineers' representations of rooms, and the main reason for using them is that they have other predefined properties than for architectural rooms. Spaces can further be used to subdivide a room. From a BOT perspective, they are both spaces and are exported as such. This means that a model containing both rooms and spaces will have duplicate geometries at some locations, but when querying over the data this can be taken care of by including properties or filtering by the content of the URI. The design of the URIs is covered in section 3.

A level in Revit is not a physical zone with containment of and adjacencies to other zones and elements as in BOT, but even though it has no physical representation it still conceptually exists and is hence exported as such.

2.2 Relationships

A limited set of relationships have been extracted at the current stage. `bot:hasSpace` describes the spaces and rooms that are located on a level, `bot:adjacentElement` describes walls surrounding and sharing a common interface with rooms/spaces and `bot:hostsElement` describes the windows and doors that are hosted in a wall. In the further development of the exporter, also zone adjacencies/containments and relevant interfaces introduced in [4] should be exported.

2.3 Properties

Two general properties, `rdfs:label` from the Resource Description Framework (RDF) Schema[3] namespace and `rvt:guid` from a fictive Revit namespace are exported along with all the exported objects. `rvt:guid` holds the Globally Unique ID (GUID) of the Revit object and the `rdfs:label` is generated a little differently based on the type of object (see Table 1).

Table 1. Properties exported from Revit.

Revit element(s)	Revit property	OWL property	Datatype
Wall, Door, Window, Level, Space, Room	GUID	rvt:guid	xsd:string
Wall, Door, Window	Type	rdfs:label	xsd:string
Wall	Width	nir:width	cdt:length
Wall	Length	nir:length	cdt:length
Level	Name	rdfs:label	xsd:string
Room, Space	Name Number	rdfs:label	xsd:string
Room, Space	Area	nir:spaceArea	cdt:ucum
Room, Space	Volume	nir:spaceVolume	cdt:ucum

[3] https://www.w3.org/TR/rdf-schema/

As the W3C LBD-CG work on properties was at the time of writing not fully developed a set of properties in a fictive Niras[4] namespace were extracted. The prod vocabulary[5] generated from IFC4 was briefly reviewed, but none of the properties listed in Table 1 were available.

In the further development, property extraction could be handled through a mapping table, thereby leaving it up to the user to specify which vocabularies to export to.

3 System architecture

The overall system architecture enables the users to (1) generate BOT-compliant triples from a Revit model (2) Upload the BIM model and convert it for rendering in the browser (3) Upload BOT-triples to a triplestore and (4) perform SPARQL[6] queries on the triplestore and filter the 3D-view based on the result.

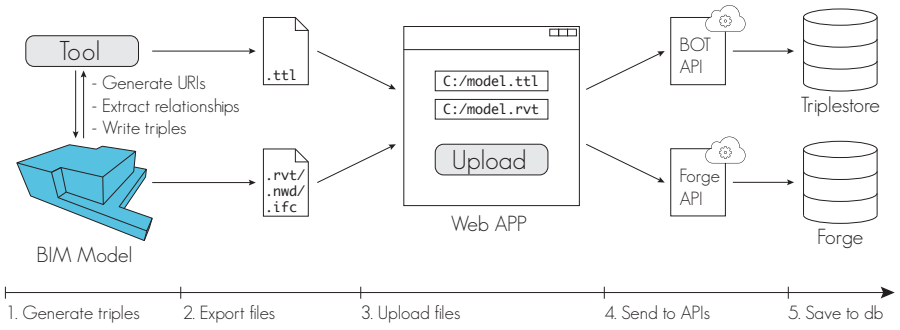

Fig. 1. The infrastructure from file export over the web application to pushing data to the two databases.

3.1 Revit

In Revit, a tool (1) generates custom property: `URI` for all objects (2) generates custom property: `host` for the project (3) generates URIs as a concatenation of `:host/:project_number/:object_type/:GUID` where the GUID is the one assigned by Revit (4) generates the BOT-relationships described in section 2 in the Resource Description Framework (RDF)[7] language and (5) exports the generated data to a Turtle[8]-file.

[4] The company in which the main author is employed.
[5] https://raw.githubusercontent.com/w3c-lbd-cg/props/master/IFC4-output.ttl
[6] https://www.w3.org/TR/sparql11-query
[7] https://www.w3.org/RDF/
[8] https://www.w3.org/TR/turtle/

Writing the URIs to a property on the Revit objects makes them available when exported to IFC or Navisworks. In the web app, these are used when filtering the 3D view. When visiting the URI in a browser the user should be presented with some useful information about the object and its relations, but at the current stage, the API is not capable of doing that.

Basing the URI on the Revit GUID establishes a mechanism for understanding the origin of an object as objects can also be created elsewhere. The URI design makes the `rvt:guid` property redundant, and it is a topic for discussion whether this property should exist.

Making the triples and the files accessible in the web app is currently handled by uploading the exported files from Revit, but it would be more user-friendly to push them directly to the web through the backend application. In Fig. 1 this would imply that step 2 and 3 could be skipped. To further improve the user experience the connection could be established through web sockets enabling a continuous communication between Revit and the server.

3.2 Web APP

Both the front- and backend of the application are built in Typescript[9] and compiled to JavaScript. JavaScript has a solid infrastructure through the Node Package Manager (npm)[10] and provides the convenience of running both on a server and in a browser. Typescript is a typed superset of JavaScript that enables better control as the application scales and further, it is the chosen language used in the Angular[11] framework which the frontend is built upon. The reason for choosing Angular is that it is developed and maintained by Google and is hence widely used and well documented. Angular further adds some guidelines for the application structure which makes it more modular and easier to scale. The BIM model is rendered in the threejs[12] based Forge Viewer[13] and Forge also handles the conversion and storage of the BIM models. Also for the triplestore a commercial product, Stardog[14], was used.

We would have preferred to base the prototype on non-commercial products like xeogl[15] on which BIMSurfer[16] is built or pure threejs for the viewer and Apache Jana Fuseki[17] for the triplestore, but the commercial solutions deliver a certainty of future maintenance and a flat learning curve since documentation is well developed.

---

[9] https://www.typescriptlang.org
[10] https://www.npmjs.com
[11] https://angular.io/
[12] https://threejs.org/
[13] https://developer.autodesk.com/en/docs/viewer/v2/overview
[14] http://www.stardog.com/
[15] https://github.com/xeolabs/xeogl
[16] http://bimsurfer.org/
[17] https://jena.apache.org/documentation/serving_data/

When adding a new project the backend creates a new database in the triplestore named P followed by the 6 digit project number (ex. `P100100`) and inserts some general information about the project in the default graph (see Lst. 1.1). Once a project is generated a so-called Bucket can be assigned to it. This can be done by clicking "add bucket" in the project overview, and doing so will generate a bucket using the Forge Data Management API and assign a new property `rvt:bucketKey` to the project in the triplestore. The bucket key must be globally unique, so in the current implementation it is niras_p followed by the project number (ex. `niras_p100100`). Buckets contain objects and objects are translated files such as BIM models. When uploading a model the file is uploaded to the backend that further sends it to the Forge API. The Forge API returns an object key and begins the translation of the file to SVF-format, which can be rendered by the viewer. The current implementation supports translations from IFC, Navisworks (NWC), Revit (RVT) and Sketchup (SKP) files but the Forge Derivative API supports +60 formats.

Listing 1.1. Project data

```
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<generatedProjectURI> a foaf:Project , doap:Project ,
                                        prov:Entity ;
        rdfs:label "projectName" ;
        doap:name "projectName" ;
        doap:created "dateTime"^^xsd:datetime ;
        rdfs:comment "projectDescription" ;
        doap:description "projectDescription" .
```

When a test file is opened in the viewer there is a significant difference to how it is rendered. The RVT-file does not contain spaces and neither does the IFC. Several elements in the IFC has deformed elements. The NWC-file includes spaces, has no deformed geometry, includes spaces and textures and has a file size of less than 10 % of the IFC and nearly 1 % of the much larger RVT-file. However, the properties of the spaces don't seem to be available from the property tree and hence it is not possible to isolate spaces based on their URI.

As the BOT semantics are stored in a separate file, this turtle-file needs to be uploaded as well. The backend receives the file and puts the triples in a named graph named `tag:/:ttlFileName:` where :ttlFileName is extracted from the received file. When more models are available, one then has the option to query based on relationships in one graph or in all graphs. Further, the named graph makes it easy to delete all triples associated with a specific model.

The triplestore can be queried from the viewer and the results are returned in a table. All results consisting of URIs are extracted and in the viewer, elements carrying these URIs are isolated as shown in Fig. 2. The example shows what wall elements are adjacent to the space, and provides a visual feedback for quality assurance and understanding purposes. Communication with Forge

and Stardog is done through two separate Express[18] based REST APIs. The one communicating with Forge just adds a middle layer for security reasons and basically extends the routs already exposed by Forge. This will not be further described. The one communicating with the triplestore handles the management of projects, upload of triples and doing queries on the graph. The full set of routes are listed in Table 2.

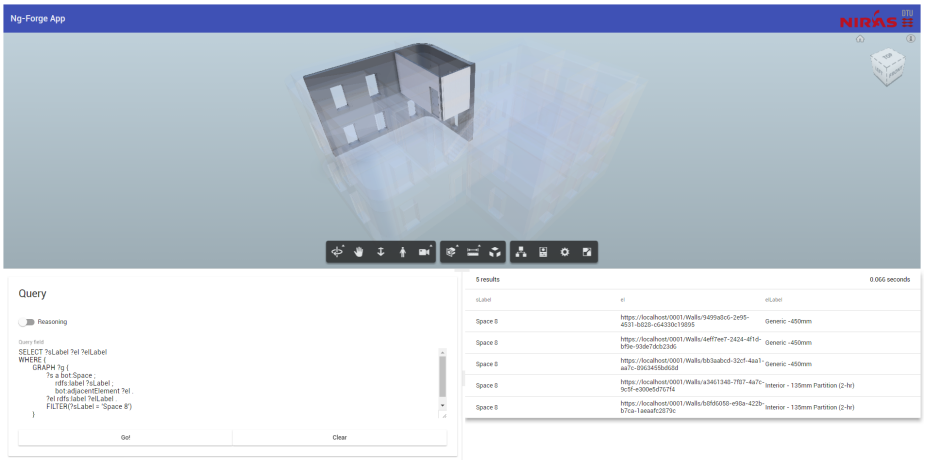

Fig. 2. Querying the model.

Get triples doesn't fully follow HTTP conventions as is it is handled by a `POST` request, but as the query is sent to the API through the request body, it is not possible to use a `GET` request.

Table 2. REST-API Routes for the backend handling communication with Stardog. :host is the address of the server that hosts the REST API (ex. https://niras.dk) and :name is the name of the database in which the requested data is stored.

Route	Method	Description
:host/projects	GET	Get all projects (databases)
:host/project	POST	Create project
:host/project/:name	DELETE	Delete project
:host/projects/:name	GET	Get project details
:host/:name/admin/postTriples	POST	Insert triples
:host/:name/admin/getTriples	POST	Get triples
:host/:name/bot	POST	Insert BOT-compliant file

[18] https://expressjs.com

4 Future work

The prototype is just a proof of concept and the list of future work is exhaustive. Some considerations were already considered in section 3, but in general we would like to see more projects and software implementations dealing with export of BIM data to BOT knowledge bases.

It would be interesting to see projects dealing with the generation of data outside the BIM tool and doing reasoning on a combination of the two. A simple example could be grouping of spaces into different zones like fire cells. It would also be interesting to see a project dealing with management of a continuous communication between the BIM tools and the triplestore. This implies dealing with properties that change over time, calculations based on properties that might change over time and the problems this might entail.

Some effort by the W3C LBD-CG is being put into developing product and property ontologies and it would be an obvious improvement of the Revit exporter to align with these. Product classes would allow for more specific queries than what is possible with `bot:Element` and there are several use cases that could benefit from having updated geometrical data from the BIM tool available at hand.

5 Conclusion

The main contribution of the study is the illustration of a simple architecture for combining 3D model data with data from a triplestore. It was succeeded to develop a working prototype of a tool to perform queries in the browser with visual 3D representations of the results, and it is the authors' belief that the visual feedback will enhance the communication of what can be expressed with BOT. At the current stage of development only some BOT data is exported from the BIM tool and this is a problem since the users might misinterpret the capabilities of the ontology.

References

1. Mads Holten Rasmussen, Pieter Pauwels, Christian Anker Hviid, and Jan Karlshøj. Proposing a central aec ontology that allows for domain specific extensions. In *Joint Conference on Computing in Construction*, volume 1, pages 237–244, 2017.
2. Thomas Liebich and Jeffrey Wix. Highlights of the development process of industry foundation classes. In *Proceedings of the 1999 CIB W78 Conference*, 1999.
3. Pieter Pauwels and Walter Terkaj. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, 63:100–133, 2016.
4. Mads Holten Rasmussen, Pieter Pauwels, Maxime Lefrançois, Georg Ferdinand Schneider, Christian Anker Hviid, and Jan Karlshøj. Recent changes in the building topology ontology. In *Linked Data in Architecture and Engineering*, 2017.