LINKING DATA WITH IFCOPENSHELL

EXTENDING IFCOPENSHELL WITH LINKED DATA TECHNIQUES

Jeroen Werbrouck
Matr. Nr. 383102

Wintersemester 2017-2018
M1 Projekt: Architekturinformatik
Supervisor: Jakob Beetz
Co-supervisors: Thomas Stachelhaus and Jyrki Oraskari

Abstract

The M1 Projekt: Architekturinformatik at the RWTH Aachen focusses on bridging the gap between architects and the software they are more than ever using. The course encourages architecture students to dive deeper into the concepts of this software, concepts that mostly remain completely hidden for the end user. Specifically, the target was to develop an extension for the 'IfcOpenShell'[1] framework, which is an open-source ifc viewer. The focus of this specific project lies in providing compatibility with some Linked Data technologies, by querying external databases as well as the project itself, nevertheless sticking to the original ifc structure. The paper discusses the added functionalities (the steps that were taken, the technologies that were used, …) and links them with current academic research. In most cases, a 'hybrid' solution was put forward to negotiate between the different technologies that are often conflicting.

Content

[1] https://github.com/IfcOpenShell/IfcOpenShell

Introduction

To enhance collaboration between the multitude of stakeholders that is typically involved in a construction project, the *Industry Foundation Classes* (ifc) form the main standard for describing, exchanging and sharing information.[2] Ifc is maintained by BuildingSMART[3] and provides an open and readable description of a Building Information Model that is (at least partly) supported by virtually all BIM-related software products. By taking the open-source IfcOpenShell[4] framework as a starting point for this project, open access to all end users is guaranteed. IfcOpenShell forms the basis of the TUEviewer (also developed at TUEindhoven): an ifc visualiser that further clarifies the hierarchical structure of the model and its properties, additonally providing a Python-compatible coding widget. In turn, the TUEviewer forms the basis for this M1 Project: it provides the necessary framework for people with no programming background at all; parallel with learning Python, an extension is developed that is more than a simple 'HelloWorld' script and can hold actual value for other people, whether applied in practice, education or academic research. To provide access to the initial IfcOpenShell files as well as to the project-specific adaptations, a Github repository has been created (https://github.com/JWerbrouck/RWTH_M1_Projekt).[5]

To keep in line with the process of learning Python, the basic goals of this project started simple and then became more complementary with current research topics (in this case mainly the relationship between Linked Data and the Industry Foundation Classes). Therefore, the first goal was to be able to change the already defined properties of a selected element, followed by the possibility to enrichen the ifc model with custom-defined propertysets. The next step introduces Linked Data and provides functionality for querying specific graphs, thereby mapping the query results to a specified element. As a final goal, the same querying method is used to query the model itself and visualising geometry that applies to the predefined restrictions.

The first objectives (those concerning only ifc and element properties and have nothing to do with Linked Data at all) are mainly a practical implementation of ifc; they are considered 'preparation steps' and don't have such an academical value. Nevertheless, they form the background of the project and could be used as a practical tool or for educational purposes. The other goals of this project (those related with Linked Data), however, do relate with current academic groundwork: currently, a lot of research is put in deepening the use of Linked Data technology in construction-related activities. Hence, alongside the discussion of the project's process and workflow, an important accent of this paper will be to relate the

[2] http://www.ifcwiki.org/index.php?title=IFC_Wiki
[3] http://www.buildingsmart-tech.org/
[4] Developed by Thomas Krijnen, TUEindhoven, http://www.ifcopenshell.org/
[5] Note: apart from adding the packages rdflib and Sparqlwrapper, all changes were made in the 'app.py' file (location: Lib/site-packages/IfcOpenShell/geom). The original application can be downloaded at https://github.com/jakob-beetz/IfcOpenShellScriptingTutorial

implemented Linked Data tools with current topics of interest. The paper ends with some short suggestions for future research.

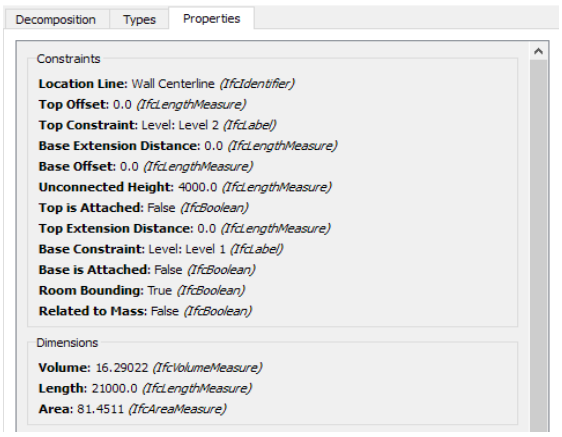
The underlying paper assumes some basic knowledge about the principles of Linked Data. Although some concepts are introduced very briefly, a general introduction to Linked Data can be found in *(Allemang and Hendler, 2011 [1])* and *(Segaran et al., 2009 [14])*. *(Pauwels et al., 2017 [12])* provides an overview of Linked Data applications in the construction industry.

1. Preparation: adapting IfcProperties

Since the project is focussing on adapting and defining properties, implementing functionality for changing an object's properties is a logical thing to start with. 'Vanilla' IfcOpenShell provides a tab widget which displays the properties just for checking purposes (Figure 1), as it initially is just an ifc viewer. Most of the functionality that is going to be added in this project, will be provided within this tab widget.

The layout of the tab widget will stay more or less the same as in the original IfcOpenShell. However, every property is now provided with an additional "QLineWidget" that allows the user to change its value. This can be done by pressing the 'return' key after typing in the desired value. Figure 2 shows the widget's current layout.

It should be noted that the application currently provides no correction or warning for nonvalid inputs (e.g. a Boolean is either 'True' or 'False', but there is no correction if the user enters something different). The user thus has to be very careful when entering new values. Only when clicking 'Save' in the 'File' menu (alternatively by pressing 'ctrl+s'), the actual changes happen by writing them to an external file. In the meantime, every entered value can still be changed. Note that returning to the original value of the property requires this value to be entered again in the textfield.



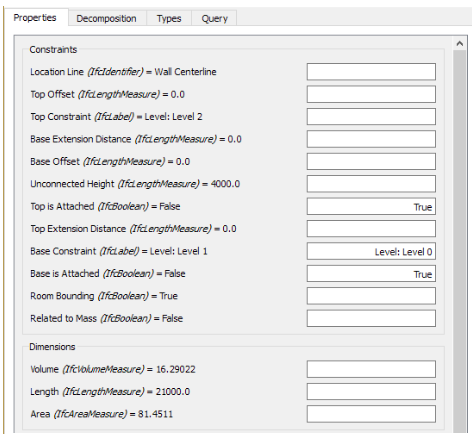
Constraints

Location Line: Wall Centerline *(IfcIdentifier)*
Top Offset: 0.0 *(IfcLengthMeasure)*
Top Constraint: Level: Level 2 *(IfcLabel)*
Base Extension Distance: 0.0 *(IfcLengthMeasure)*
Base Offset: 0.0 *(IfcLengthMeasure)*
Unconnected Height: 4000.0 *(IfcLengthMeasure)*
Top is Attached: False *(IfcBoolean)*
Top Extension Distance: 0.0 *(IfcLengthMeasure)*
Base Constraint: Level: Level 1 *(IfcLabel)*
Base is Attached: False *(IfcBoolean)*
Room Bounding: True *(IfcBoolean)*
Related to Mass: False *(IfcBoolean)*

Dimensions

Volume: 16.29022 *(IfcVolumeMeasure)*
Length: 21000.0 *(IfcLengthMeasure)*
Area: 81.4511 *(IfcAreaMeasure)*

Figure 2: tab:properties (as-is)



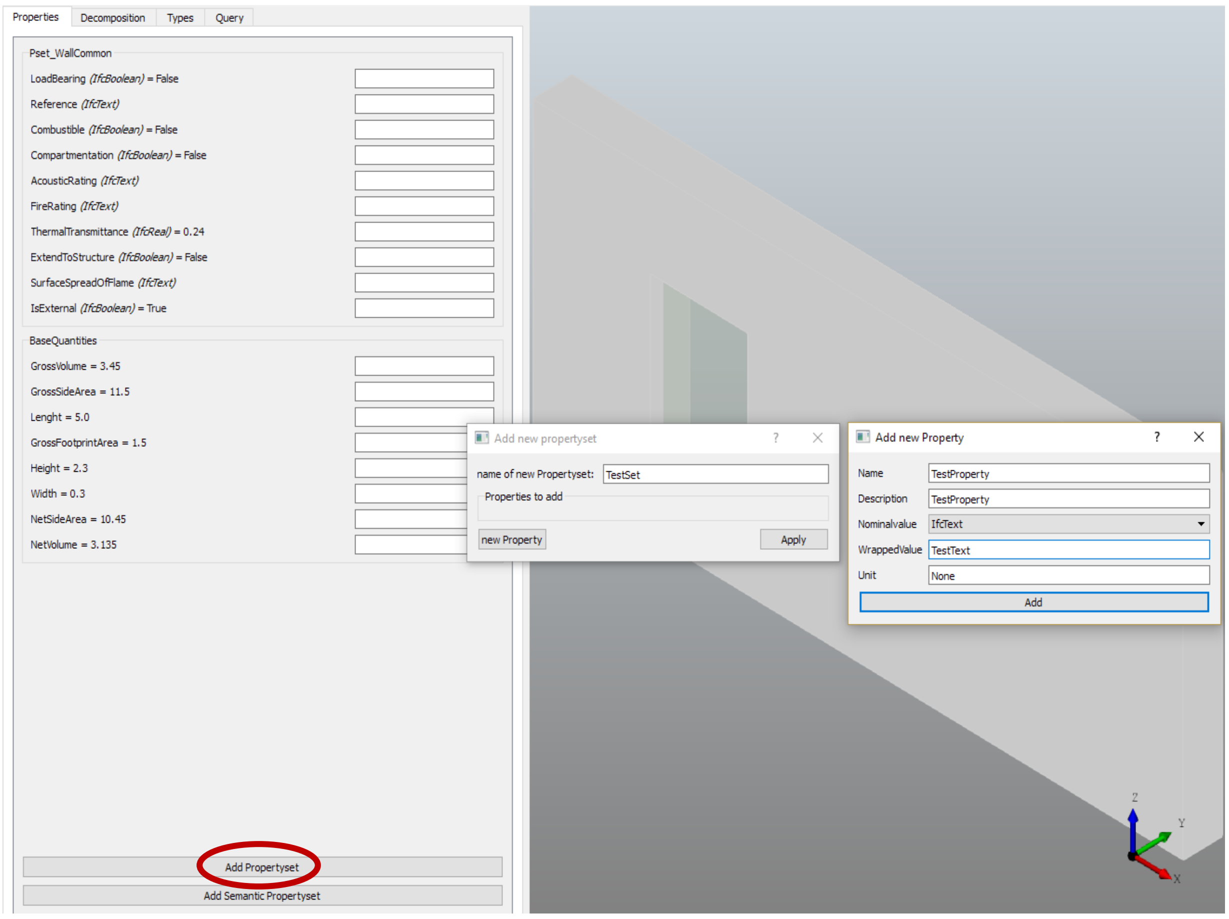
Constraints	
Location Line *(IfcIdentifier)* = Wall Centerline	
Top Offset *(IfcLengthMeasure)* = 0.0	
Top Constraint *(IfcLabel)* = Level: Level 2	
Base Extension Distance *(IfcLengthMeasure)* = 0.0	
Base Offset *(IfcLengthMeasure)* = 0.0	
Unconnected Height *(IfcLengthMeasure)* = 4000.0	
Top is Attached *(IfcBoolean)* = False	True
Top Extension Distance *(IfcLengthMeasure)* = 0.0	
Base Constraint *(IfcLabel)* = Level: Level 1	Level: Level 0
Base is Attached *(IfcBoolean)* = False	True
Room Bounding *(IfcBoolean)* = True	
Related to Mass *(IfcBoolean)* = False	
Dimensions	
Volume *(IfcVolumeMeasure)* = 16.29022	
Length *(IfcLengthMeasure)* = 21000.0	
Area *(IfcAreaMeasure)* = 81.4511	

Figure 1: tab:properties (plugin)

2. Preparation: defining custom IfcPropertySets

The next basic functionality that needs to be implemented before starting the Linked Data chapter is to provide a way to define custom propertysets. This also happens within the *tab:properties*, but this time as a pop-up window with an overview of the added properties and the name of the "*IfcPropertySet"*, and another pop-up window for defining the new properties (Figure 3). These widgets are called by pressing the 'Add Propertyset' button at the bottom of the *tab:properties* widget. As was also the case in the previous section, there is no automatic correction for integers, doubles or Booleans, so, again, maintaining good concentration is quintessence. Further on, the user should take care in defining a unique name for the new propertyset. Although this is not necessary for ifc in general, the name of the propertyset will be used later on to prevent the software from writing the same propertyset multiple times to the same file (when saving repeatedly within the same work session).

When defined, a new *IfcPropertySet* is not immediately written away; instead, it is stored in a temporary list. This list can be checked through the 'Extra' button on the menu bar (Figure 4).



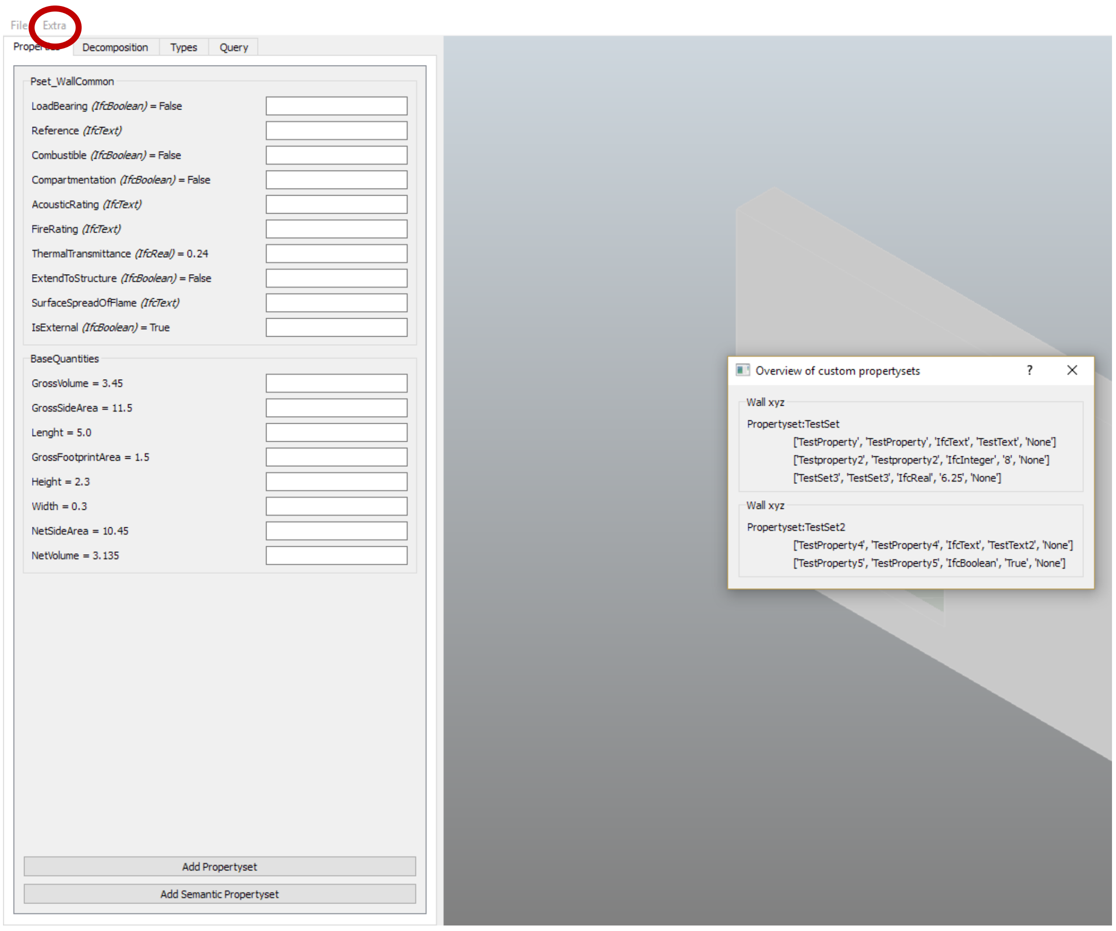
Pset_WallCommon
LoadBearing *(IfcBoolean)* = False
Reference *(IfcText)*
Combustible *(IfcBoolean)* = False
Compartmentation *(IfcBoolean)* = False
AcousticRating *(IfcText)*
FireRating *(IfcText)*
ThermalTransmittance *(IfcReal)* = 0.24
ExtendToStructure *(IfcBoolean)* = False
SurfaceSpreadOfFlame *(IfcText)*
IsExternal *(IfcBoolean)* = True

BaseQuantities
GrossVolume = 3.45
GrossSideArea = 11.5
Lenght = 5.0
GrossFootprintArea = 1.5
Height = 2.3
Width = 0.3
NetSideArea = 10.45
NetVolume = 3.135

name of new Propertyset: TestSet
Properties to add

Name	TestProperty
Description	TestProperty
Nominalvalue	IfcText
WrappedValue	TestText
Unit	None

Figure 3: adding a custom IfcProperty(Set)



Pset_WallCommon

LoadBearing *(IfcBoolean)* = False
Reference *(IfcText)*
Combustible *(IfcBoolean)* = False
Compartmentation *(IfcBoolean)* = False
AcousticRating *(IfcText)*
FireRating *(IfcText)*
ThermalTransmittance *(IfcReal)* = 0.24
ExtendToStructure *(IfcBoolean)* = False
SurfaceSpreadOfFlame *(IfcText)*
IsExternal *(IfcBoolean)* = True

BaseQuantities

GrossVolume = 3.45
GrossSideArea = 11.5
Lenght = 5.0
GrossFootprintArea = 1.5
Height = 2.3
Width = 0.3
NetSideArea = 10.45
NetVolume = 3.135

```
Wall xyz
Propertyset:TestSet
            ['TestProperty', 'TestProperty', 'IfcText', 'TestText', 'None']
            ['Testproperty2', 'Testproperty2', 'IfcInteger', '8', 'None']
            ['TestSet3', 'TestSet3', 'IfcReal', '6.25', 'None']
Wall xyz
Propertyset:TestSet2
            ['TestProperty4', 'TestProperty4', 'IfcText', 'TestText2', 'None']
            ['TestProperty5', 'TestProperty5', 'IfcBoolean', 'True', 'None']
```

Figure 4: overview of added propertysets

3. Linked Data and BIM

In the previous sections, the focus has been on exploring the basics of ifc in developing some functionality that makes the TUEviewer more than a mere viewer. Although they make IfcOpenShell usable to quickly change or add properties, these adaptations stayed strictly within the ifc framework. They did not engage in or relate to academic research going on in the field of Building Information Modeling. The next part of the paper discusses the second stage of the project, which goes beyond ifc and introduces some more exotic topics, such as graphs, ontologies, SPARQL … all of which can be grouped under 'Linked Data'.

One of the main challenges here will be to connect the syntax of a non-semantic IfcOpenShell with the triple-based graphs that are essential for Linked Data. After all, IfcOpenShell can only open 'classic' EXPRESS-based ifc files and is not meant to work with triples or ontologies (which serve to describe building elements and their relationships in a "Linked Data" way). Yet, as we will see, this duality does not have to be bad at all, since it can help in filling the gap between academic research (concerning Linked Data) and everyday practice (just now starting to adopt Linked Data technologies). Within certain limits, results of linked data can be coupled with ‘classic’ ifc files, providing some gradual transition towards a more severe implementation of Linked Data technologies in the AEC industry, which holds a lot of promising prospects.[6]

In general, improving BIM with Linked Data technologies is regarded as the ‘next step’ in further enriching and linking building models with external information or disciplines (e.g. GIS or heritage), rule checking, extraction of ‘submodels’, … *(Beetz et al., 2016 [2]; Beetz et al., 2014 [3]; Costa et al., 2015 [4]; Pauwels et al., 2010 [7]; Pauwels et al., 2011 [11]; Pauwels et al., 2017 [12])*. For this project’s case, three aspects are considered more relevant than the others: querying specific product catalogues and link the results to a component *(Costa et al., 2015 [4])*, linking information from disciplines that are adjacent to BIM (e.g. GIS or information regarding built heritage) *(Beetz et al., 2016 [2]; 2014 [3])* and the inherent data structure that allows to search the model itself with very specific queries. The next section will discuss the first two and relate them to the project’s ‘hybrid’ solution, the subsequent part will do the same for the last one.

4. Linked Data: querying external databases

Because of this project’s hybrid nature, the more exotic benefits of Linked Data stay out of scope. There are, however, some use cases that allow for a connection with applications that are initially not related with Linked Data, such as IfcOpenShell. One of these cases is the ability to query databases for very specific results, using SPARQL[7] (“Simple Protocol And RDF Query Language”) as the querying language. Having the possibility to query external databases and link the results to a building element opens some new doors for specifying product information or linking interdisciplinary (meta)data…

Product Catalogues

A logical example for combining Linked Data and BIM is the ability to search product catalogues for specific entities. For some software packages, such catalogue databases already exist and are either the initiative of the product manufacturers themselves or third parties that provide databases with products of different companies. An example for BIM in general is BIMobject.com[8], but there are also databases for very specific applications (e.g. for Lighting Analysis), such as LumSearch[9] for DIALux. The main disadvantage of these databases is that the information they contain is mostly very software-specific. When you take a look at the jungle of different formats present in Figure 5, this becomes painfully clear. It would save a lot of work and money when manufacturers or third parties would be able to implement products (and corresponding (meta)data) in a standardized procedure that can be used by all BIM-

[6] An overview of advantages in different subdomains is given in *(Pauwels et al., 2017 [12]). (Pauwels et al., 2015 [10])* bundles some papers that share the topic of enhancing BIM with Linked Data.
[7] https://www.w3.org/TR/sparql11-query/
[8] http://bimobject.com/en-us (BIMobject is the successor of Autodesk SEEK, which combines different kinds of BIM-families from different manufacturers)
[9] http://lumsearch.com/en-US#0

packages. This standardized procedure can be achieved by specifying data through RDF[10] (“Resource Description Framework”), which links information by usage of triples (subject - predicate - object), thus forming a connected graph full of nodes and relationships. RDF forms the basic ‘language’ of the semantic web and is the worldwide standard for describing, connecting and distributing information in the semantic web. More info regarding the syntax of RDF and its extensions can be found in [1] and [14].

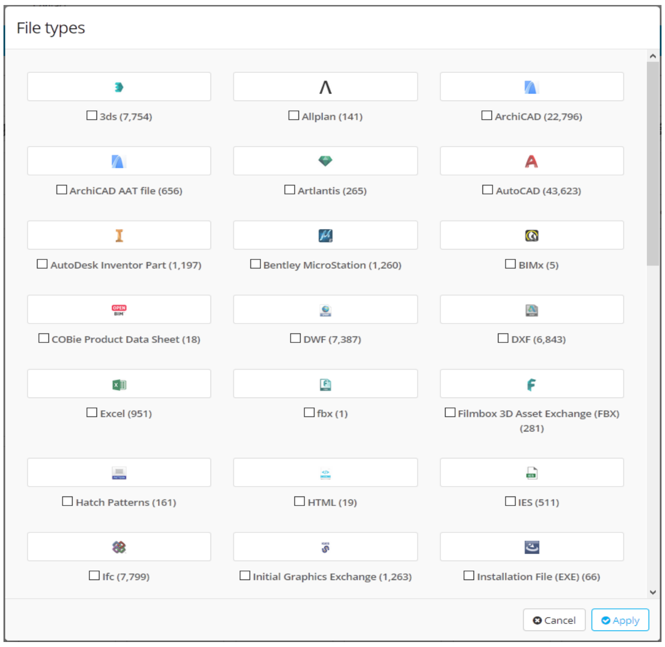
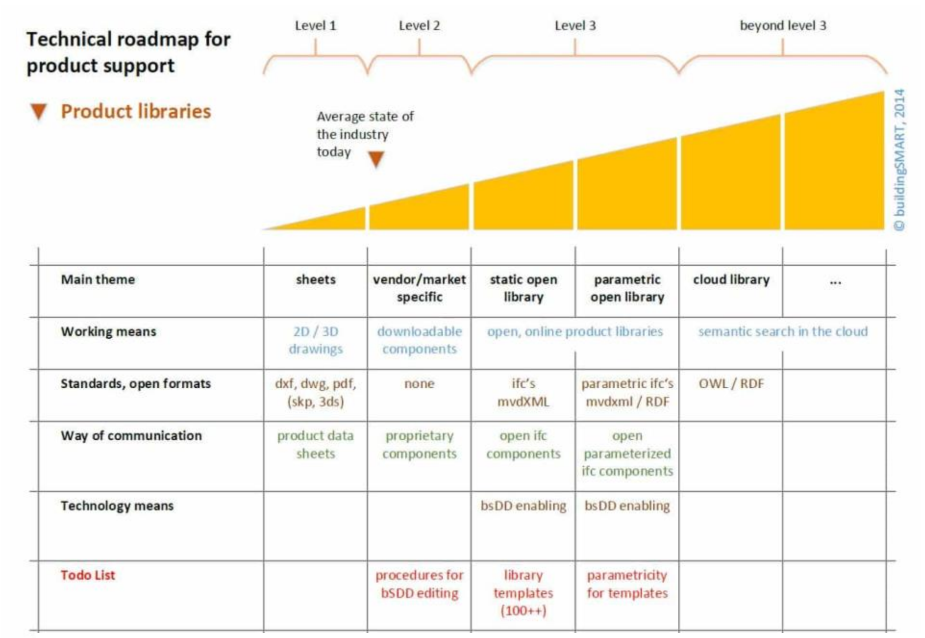


Figure 5: querying for products for different applications. (source: http://bimobject.com/en-us/product)

Some initiatives have been made for providing such triple-based building product databases. An important one is the bsDD[11] (BuildingSMART Data Dictionary), which serves as a foundation for instance libraries. The bsDD defines general terms that can, for example, be used for product manufacturers to specify information about their products. Further on, its aim is to be language independent, thus being usable worldwide. The bsDD was extensively used for code-testing and as a guide for decision-taking concerning functionality and user interface in different stages of the project. A product catalogue that bundles specific instances for products from different European manufacturers has been constructed in context of BauDataWeb[12].

According to Figure 6, in 2014, the state of the industry regarding product support lied at the very beginning of ‘level 2’, which means vendor or market specific product libraries with downloadable components, but without standardisation. Nowadays (2018) this state has probably shifted towards somewhere at the beginning of level 3 (this is the author’s assumption, since no up-to-date schemes were found). As indicated in Figure 6, level 3 is just the beginning of open, online product libraries, with bsDD support. While in academic environments, lots of research are being carried out for deepening the implementation of Semantic Web technologies for the AEC industry, everyday industry praxis reaching level ‘beyond level 3’ is not something that will be achieved in the next couple of years.

[10] https://www.w3.org/RDF/
[11] https://www.buildingsmart.org/standards/standards-tools-services/data-dictionary/
[12] http://www.freeclass.eu/semanticSearch_Information


Technical roadmap for product support
Product libraries
Average state of the industry today
© buildingSMART, 2014

	Level 1	Level 2	Level 3		beyond level 3	
Main theme	sheets	vendor/market specific	static open library	parametric open library	cloud library	...
Working means	2D / 3D drawings	downloadable components	open, online product libraries		semantic search in the cloud	
Standards, open formats	dxf, dwg, pdf, (skp, 3ds)	none	ifc's mvdXML	parametric ifc's mvdxml / RDF	OWL / RDF	
Way of communication	product data sheets	proprietary components	open ifc components	open parameterized ifc components		
Technology means			bsDD enabling	bsDD enabling		
Todo List		procedures for bSDD editing	library templates (100++)	parametricity for templates		

Figure 6: Technical roadmap Product Libraries - state of the industry in 2014 (source: https://www.buildingsmart.org/standards/technical-vision/technical-roadmaps/)

Interdisciplinary Linking

The discussion about product catalogues can be generalized to implement information that is not directly related to BIM. Databases containing information about e.g. GIS or heritage can be queried as well, the results being linked to the currently selected element. As indicated in [12], linking across domains and interdisciplinarity can be considered as main arguments for the AEC industry to adopt Linked Data technologies.

Most disciplines have their own way of describing relevant information. This is great when operating within the domain itself, but there is not a single discipline that can be considered isolated. On the contrary; many disciplines share similar topics, often only differing in their main research focus. Bridging the gap between these frequently occurring interdisciplinary overlaps and the domain-specific ways of describing things can be done by adopting Linked Data techniques, which then serve as an infrastructure that provides common ground for data exchange (a parallel with the different product extensions from the previous paragraph can be drawn here). Nevertheless, even when adopting this infrastructure, each discipline keeps having its own rules and limitations. These rules and internal relationships are defined in domain-specific 'ontologies'. Apart from stating internal relationships, the use of ontologies also helps to integrate data coming from different domains and mediate between possibly contradictory definitions (e.g. when the same word is used to describe a different thing, or different words are used to describe the same thing).

For example, to couple specific historical information to building elements, you need to combine geometrical with historical data. Both have different areas of interest but meet each other in several topics, such as history of architecture, built heritage or renovation projects. A way of describing building data is by using ifcOWL *(Pauwels and Terkaj, 2016 [9])* or BOT *(Rasmussen et al., 2017 [13])* ontologies, whereas other ontologies exist for description of cultural heritage data, such as the CIDOC-CRM ontology *(Le Boeuf et al., 2017 [6])*. An example for an intermediating ontology is CHML ("Cultural Heritage Markup Language") *(Hauck et al., 2015, [5])*, an ontology based on CIDOC-CRM that allows to link historical data and geometry. The building's geometry could be extracted from the BIM graph[13] and then related to the specific historical information, or the mapping could happen directly between graphs.

This provides the background for this part of the project; a background for which the requirements, however, are not met, since we are using EXPRESS-based ifc, which has nothing to do with RDF. The next paragraph describes the hybrid method that is applied in the project to work around this 'issue'. The method for linking data from product catalogues will be the same as for linking information from other disciplines.

Implementation

As has been indicated previously, one of the main goals of this project is to bring research and industry closer together regarding this topic. Although IfcOpenShell can only interpret 'default', EXPRESS-based ifc files, the added functionality makes it possible to couple this with Linked Data queries, staying within the classic framework but reaching out to new technology and illustrating possible opportunities. This coupling is done by use of *rdflib*[14] and *Sparqlwrapper*[15], two Python packages for adding Linked Data features to the application.

As was also the case with the previous functionalities (property adaptation and definition), this widget can be called in the *tab:properties*, by pressing the button "Add Semantic Propertyset". This opens a pop-up window (Figure 7), where the process starts by specifying the URI of the graph (or the path to the local folder where the graph is located). Then, two options are provided: the first one provides just a rough method to query for words contained in the label of the resource (meant for people who are not familiar with constructing SPARQL queries), the second one provides full SPARQL compatibility and serves thus as a general SPARQL endpoint for querying graphs.

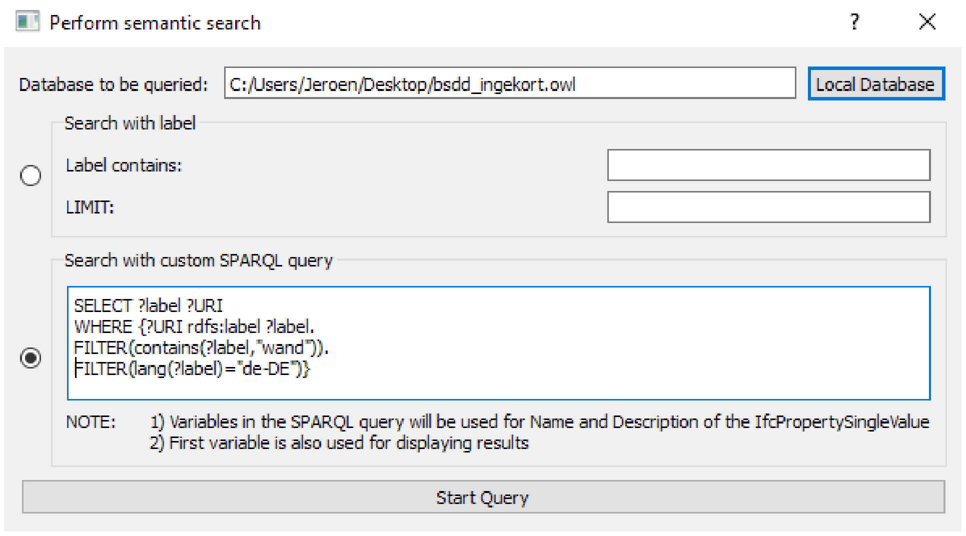
The Database that is used in this example, is the bsDD. A very simple SPARQL query was constructed (the example queries for the label and the URI from all elements for which the label contains the word 'wand', only selecting 'German' labels). Note that the semantics of

[13] NOTE: currently, there is no 'best solution' to describe geometrical data with RDF. Geometrical data is typically cumbersome and very difficult to implement efficiently. Different approaches exist alongside each other, each with its own (dis)advantages (source: Pauwels et al., 2017 [8])

[14] https://rdflib.readthedocs.io/en/stable/

[15] https://rdflib.github.io/sparqlwrapper/

‘?label’ and ‘?URI’ are only defined by the relationships stated in the query (another ‘name’ for the variable would function just as well).


Perform semantic search

Database to be queried: C:/Users/Jeroen/Desktop/bsdd_ingekort.owl

Search with label

Label contains:

LIMIT:

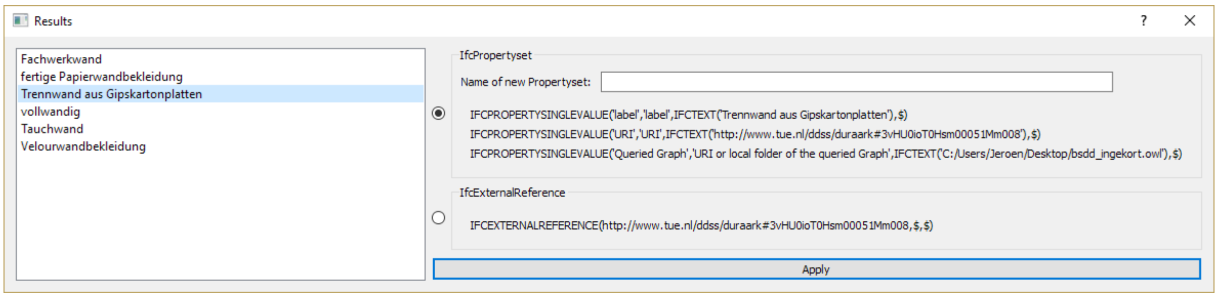
Search with custom SPARQL query

```
SELECT ?label ?URI
WHERE {?URI rdfs:label ?label.
FILTER(contains(?label,"wand")).
FILTER(lang(?label)="de-DE")}
```

NOTE: 1) Variables in the SPARQL query will be used for Name and Description of the IfcPropertySingleValue
2) First variable is also used for displaying results

Figure 7: Database selection and SPARQL query

When clicking the “Start Query” button, the programme first checks if this graph has already been parsed, to avoid double work. This can mean significant time gains, since in the current version, parsing is done locally (at desktop level). Parsing a huge graph combined with using old hardware can take a lot of time and should be avoided when unnecessary. A future version may implement this into a web viewer, leaving the parsing to a specialized server that can do the job in an eye blink. Once the graph has been parsed, the software tries to query it with the constructed SPARQL query. If the query contains syntax errors, the process is terminated; if the SPARQL syntax is correct, a new window pops up (Figure 8), showing the result(s). If there are multiple results, the user can select the correct one and hit apply. Again, two options are provided: in the first one, the results are linked to the selected element as an “*IfcPropertyset*”, the second one links the URI of the selected result to the file as a whole, by means of an “*IfcExternalReference*”.


Results

Fachwerkwand
fertige Papierwandbekleidung
Trennwand aus Gipskartonplatten
vollwandig
Tauchwand
Velourwandbekleidung

IfcPropertyset

Name of new Propertyset:

```
IFCPROPERTYSINGLEVALUE('label','label',IFCTEXT('Trennwand aus Gipskartonplatten'),$)
IFCPROPERTYSINGLEVALUE('URI','URI',IFCTEXT('http://www.tue.nl/ddss/duraark#3vHU0ioT0Hsm00051Mm008'),$)
IFCPROPERTYSINGLEVALUE('Queried Graph','URI or local folder of the queried Graph',IFCTEXT('C:/Users/Jeroen/Desktop/bsdd_ingekort.owl'),$)
```

IfcExternalReference

```
IFCEXTERNALREFERENCE(http://www.tue.nl/ddss/duraark#3vHU0ioT0Hsm00051Mm008,$,$)
```

Figure 8: displaying the results of the SPARQL query

Because EXPRESS ifc was initially not meant to be combined with Linked Data technologies, using an "*IfcPropertySet"* to link the results to the element is more an ad-hoc way to achieve the goal and is certainly not a widely accepted standard. Each variable that is defined in the SPARQL query ("SELECT ?… ?... ") gets assigned to an individual property. The method is partly subjective, since the variables are used for the name and description of the property. The wrapped value, which should be (and is) entirely unambiguous, corresponds with the value of the item that is currently selected in the displayed list; it is not influenced by the way the SPARQL query is formulated. In a normal SPARQL query, the order of variables has no influence on the results. This remains the case here, but the plugin uses the first assigned variable for displaying, so if the user is aware of this, it can help for quicky seeing through the results (e.g. if the first variable refers to the resource's label, these labels are displayed in the list at the left of the widget). Finally, an extra property is constructed to refer to the URI (or local folder) where the graph is located. The *IfcPropertySet* groups all these properties and is then connected with the element that is currently selected in the viewer.

The ifc lexicon also contains an entity called "*IfcExternalReference*", which "*is the identification of information that is not explicitly represented in the current model or in the project database*".[16] This is, in fact, exactly what we're looking for. However, the method of connecting this reference to an actual element is not present yet in IfcOpenShell (which is based on IFC2x3). Instead, the link is added to the ifc file as a whole. This functionality is, however, present in IFC4, which is currently the newest version but is not (yet) supported by IfcOpenShell. Nonetheless, if this "*IfcExternalReferenceRelationship*"[17] would be implemented in the near future: the source code contains a (currently disabled) snippet that, when enabled, should fix this issue and make this the preferable way of linking external information to elements and save it to a new file.

Following one of these methods, the adapted files stay compatible with existing software packages that do not implement RDF-based model definitions yet, while the content of the file is kept aligned with current tendencies to take part in the 'next world wide web', by referencing external sources through their URI. The next section focusses on the third application discussed in section "Linked Data and BIM", which is entirely different, namely taking advantage of the inherent data structure that allows to search the model itself with very specific queries.

[16] http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcexternalreferenceresource/lexical/ifcexternalreference.htm
[17] http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcexternalreferenceresource/lexical/ifcexternalreferencerelationship.htm

5. Linked Data: querying the model internally

Even when considering the ifc file isolated from external databases, applying Linked Data techniques still holds benefits. For example; apart from the coupling of data, an inherent function of Linked Data is that there is virtually no limit to the detailedness of description of an entity. Applied to buildings, this means that every possible detail of the project can (but doesn't necessarily has to) be specified, in contrast to EXPRESS ifc, which only has a fixed number of classes and relationships. For example, instead of just stating that something is an '*IfcDoor*', an architect could specify the exact type of door knob, hinges, … being able to design further and further until the desired level of detail is reached.

Nowadays, there are several ontologies that take care of this "translation" from EXPRESS ifc to 'ontology' ifc. One example is ifcOWL *(Pauwels and Terkaj, 2016* [9]), which is an almost literal ifc equivalent, but then formulated in a Linked Data ontology (and not in EXPRESS). IfcOWL also became an official standard for ifc description in IFC4. Another example of a building ontology is BOT (Building Topology Ontology) *(Rasmussen et al., 2017* [13]), which is lighter, more manageable and less complicated than ifcOWL. Ontology-based BIMs can be queried just like any other RDF-based graph, what allows to search the file very thoroughly. Implementing this characteristic in an ifc viewer, in this case IfcOpenShell, could help in highlighting certain elements; only those elements that apply to the rules defined in a very detailed SPARQL query. To the end user, this can be valuable for checking purposes (*"show me the walls that have a U-value higher than 0.3 W/m²K"*) or serve as an aid to semantically enrichen specific building elements. For example, one might want to say that the marble of each marble-based ground-level floor originates from a Carrara quarry. The first step is visualising marble-based ground-level floors through a SPARQL query. The next step could be selecting these elements and 'enrichen' them with information about the quarry (e.g. URI of the DBPedia page of the quarry, georeferences, …).

Implementation

This project aims to widen the EXPRESS-based IfcOpenShell by introducing Linked Data concepts, without handing in on compatibility with other ifc supporting applications or ease of use. The solution that is proposed to be able to query the model semantically, despite its EXPRESS syntax, is the coupling of the original file with its translation to an RDF graph. For this, a separate tab widget is provided (*tab:query*). The workflow is demonstrated in Figures 9-10 and takes place as follows:

1. Converting ifc to RDF

The first step is to convert the original ifc file to an RDF graph. This can be done at the following repository: *http://smartlab1.elis.ugent.be:8889/IFC-repo/*. After the file has been uploaded, it is converted to the Turtle format (*.ttl), a graph representation format that is also quite readable for humans. After the conversion is done, it can be downloaded for further use.

2. Specifying the path to the .ttl file

By clicking the […] button (top left), a loading widget will be opened for selecting the correct graph (i.e. the exact translation of the opened file into triples). Alternatively, the path could be typed in manually.

3. OPTIONAL: pre-parsing the graph

To serve as an aid for constructing SPARQL queries, there is an option to pre-parse the graph, displaying lists with subjects, predicates and objects that are present in the graph. Clicking on a specific item in one of these lists inserts this item in the text field, reducing the chance for typos. If this option is not used, the graph will be parsed in a later stage, when clicking the "Search" button.

4. Defining the SPARQL query

The uppermost text field serves as an input for SPARQL queries. The user can construct the query all by himself or use the pre-parsing functionality as an aid. For this demonstration, a very simple query is constructed, just querying for all the elements in the model that are an "*IfcWallStandardCase*":

$$SELECT\ ?wall\ WHERE\ \{?wall\ rdf{:}type\ ifcowl{:}IfcWallStandardCase\}$$

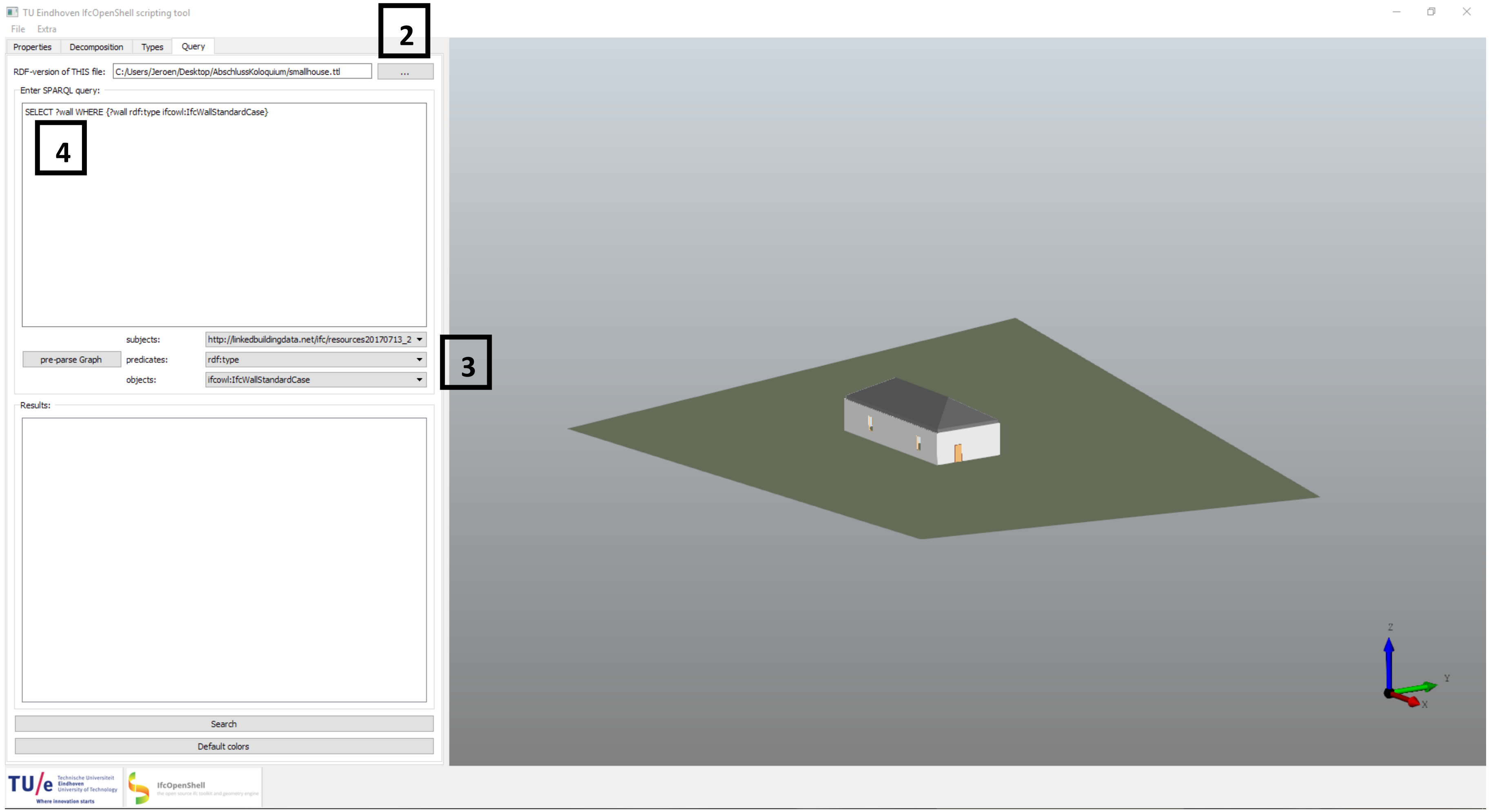
5. (Parsing), querying and displaying the results

If step 3 is skipped, hitting the "Search" button parses the graph. Then, the SPARQL query is used to search for matches. Finally, the matching results are displayed in the widget 'results' and (if they are geometric) highlighted in the viewer (in fact, all elements but the matches are displayed in wireframe-mode, the matches remaining clearly visible). Selecting one of the results in the list widget gives the corresponding element a red colour in the viewer.

It turns out that all the walls in this project of the type: *IfcWallStandardCase*, since all present walls remain visible (Figure 10).

6. Resetting the default colors

To reset the element colors back to default, the user can press the "Default colors" button. The 'default colors' are the colors as defined in IfcOpenShell, and are not necessary identical to the files' colors (Figure 9).



RDF-version of THIS file: C:/Users/Jeroen/Desktop/AbschlussKoloquium/smallhouse.ttl

Enter SPARQL query:

```
SELECT ?wall WHERE {?wall rdf:type ifcowl:IfcWallStandardCase}
```

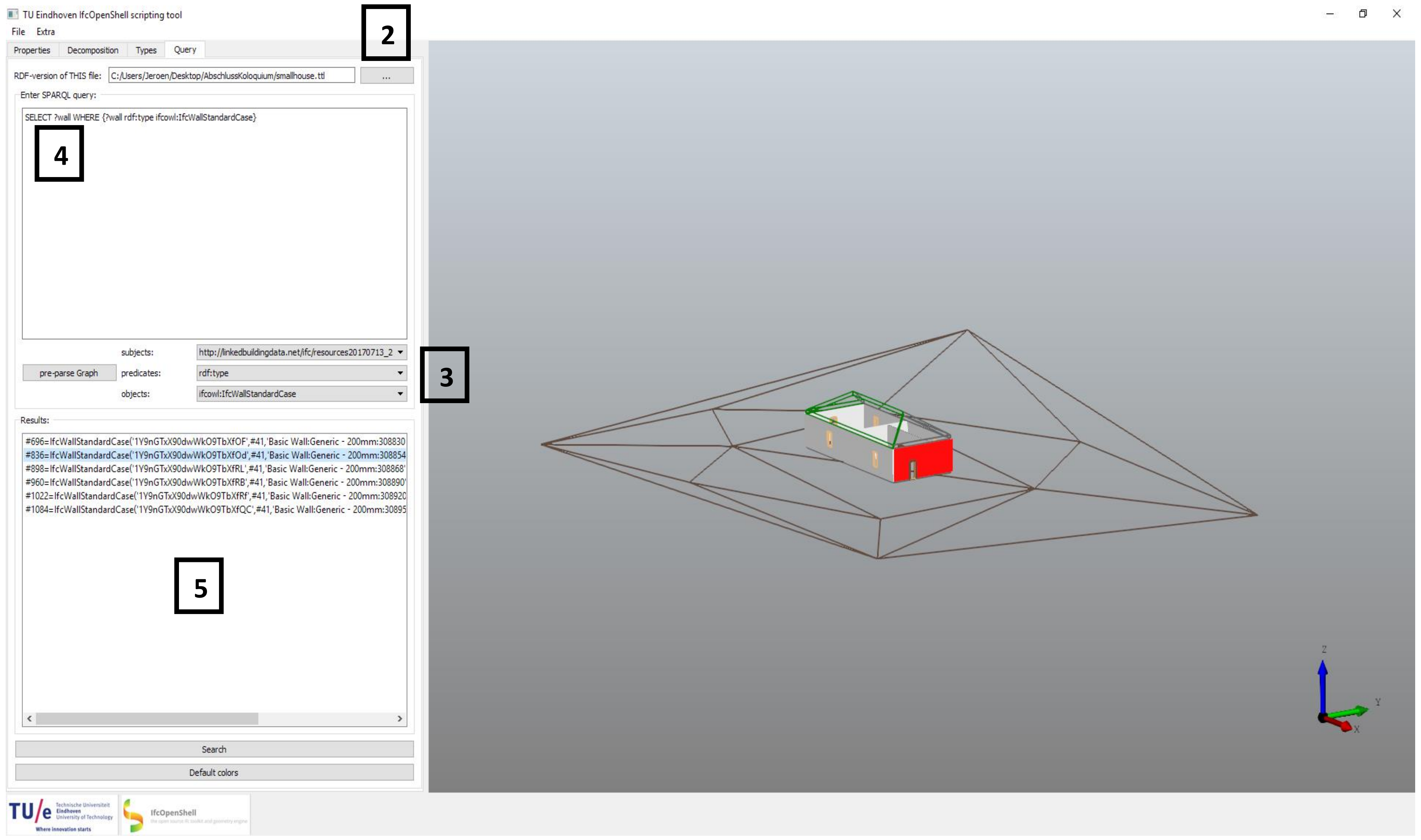
subjects: http://linkedbuildingdata.net/ifc/resources20170713_2

predicates: rdf:type

objects: ifcowl:IfcWallStandardCase

Results:

Figure 9: Interface for internal querying – preview BEFORE QUERY



RDF-version of THIS file: C:/Users/Jeroen/Desktop/AbschlussKoloquium/smallhouse.ttl

Enter SPARQL query:

```
SELECT ?wall WHERE {?wall rdf:type ifcowl:IfcWallStandardCase}
```

subjects: http://linkedbuildingdata.net/ifc/resources20170713_2

predicates: rdf:type

objects: ifcowl:IfcWallStandardCase

Results:

```
#696=IfcWallStandardCase('1Y9nGTxX90dwWkO9TbXfOF',#41,'Basic Wall:Generic - 200mm:308830
#836=IfcWallStandardCase('1Y9nGTxX90dwWkO9TbXfOd',#41,'Basic Wall:Generic - 200mm:308854
#898=IfcWallStandardCase('1Y9nGTxX90dwWkO9TbXfRL',#41,'Basic Wall:Generic - 200mm:308868'
#960=IfcWallStandardCase('1Y9nGTxX90dwWkO9TbXfRB',#41,'Basic Wall:Generic - 200mm:308890'
#1022=IfcWallStandardCase('1Y9nGTxX90dwWkO9TbXfRf',#41,'Basic Wall:Generic - 200mm:308920
#1084=IfcWallStandardCase('1Y9nGTxX90dwWkO9TbXfQC',#41,'Basic Wall:Generic - 200mm:30895
```

Figure 10: Interface for internal querying – AFTER QUERY: visualisation

Further research

During the course of this project, many ideas for 'extra' functionality popped up, of which there were only a few that were developed further. However, some of the ideas that did not make it into this project, might be interesting to implement into a future version. A few suggestions are made that would fit into the general topics of this project.

- Implementing functionality to extract Model View Definitions (MVD's) from the 'Internal query': specific queries allow to isolate certain elements (e.g. all elements concerning HVAC equipment) then export them into a new ifc-file that only contains the relevant aspects.

- The ability to only import geometry (without ifc information), assigning semantics and relationships from different sources, guided by ontologies such as ifcOWL, BOT or CHML; the geometry being the 'glue' that connects these semantics. This could be framed in the context of modeling as-built BIMs: starting with modeling only the geometry, then assigning information about sources, assumptions, uncertainties, relationships, historical data, GIS etc. This implies some 'broad BIM' interpretation: constructing a 'smart', interconnected building model.

Conclusion

This project made a connection between Linked Data and 'classic' ifc. After providing some basic functionality for property-adaptation, concepts from the academic 'next-generation BIM' (based on Linked Data) were drawn into the project to be coupled with EXPRESS based ifc. Combining these often created collisions, ad-hoc solutions or workarounds. In most cases, a middle ground was created to negotiate between these two; keeping the EXPRESS ifc scheme used by IfcOpenShell (hence also keeping compatibility with other BIM packages).

At the beginning of the project, I knew nothing about ifc, Python and Linked Data at all. Throughout the process, knowledge about these topics developed further, often clearing the fog over problems that were created in an earlier stage. This also means that the coding style evolved with the project: most of the added code is very un-Pythonic, far-fetched and ad-hoc. This way of being thrown into the deep end also makes the code unnecessary complex and almost un-editable because of nesting functions within functions, storing results in global lists and so on. However, all initial goals were reached within time, along with some other goals that rose during the project. Doing this M1 project provided me with some important skills and knowledge that will probably influence my working habits for the rest of my architectural career.

Bibliography

[1] Allemang, D., Hendler, J., 2011. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL, 2nd Edition. Morgan Kaufmann Publishers (Elsevier).

[2] Beetz, J., Blümel, I., Dietze, S., Fetahui, B., Gadiraju, U., Hecher, M., Krijnen, T., Lindlar, M., Tamke, M., Wessel, R., Yu, R., 2016. Enrichment and Preservation of Architectural Knowledge, in: 3D Research Challenges in Cultural Heritage II, Lecture Notes in Computer Science. Springer, Cham, pp. 231–255.

[3] Beetz, J., Coebergh van den Braak, W., Botter, R., Zlatanova, S., de Laat, R., 2014. Interoperable data models for infrastructural artefacts — a novel IFC extension method using RDF vocabularies exemplified with quay wall structures for harbors, in: A. Mahdavi, B. Martens, R. Scherer (Eds.), Proceedings of the 10th European Conference on Product and Process Modelling, 135-136

[4] Costa, G., Madrazo, L., 2015. Connecting building component catalogues with BIM models using semantic technologies: an application for precast concrete components. Autom. Constr. 57, 239-248

[5] Hauck, O., Kuroczynski, P., 2015. Cultural Heritage Markup Language - How to Record and Preserve 3D Assets of Digital Reconstruction. CHNT Proceedings.

[6] Le Boeuf, P., Doerr, M., Emil Ore, C., Stead, S., 2017. Definition of the CIDOC Conceptual Reference Model. ICOM/CIDOC Documentation Standards Group

[7] Pauwels, P., De Meyer, R., Van Campenhout, J., 2010. Interoperability for the Design and Construction Industry through Semantic Web Technology, in: Semantic Multimedia, 5th International Conference on Semantic and Digital Media Technologies (SAMT 2010), 143-158.

[8] Pauwels, P., Krijnen, T., Terkaj, W., Beetz, J., 2017. Enhancing the ifcOWL ontology with an alternative representation for geometric data. Autom. Constr. 80, 77–94.

[9] Pauwels, P., Terkaj, W., 2016. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. Autom. Constr. 63, 100–133.

[10] Pauwels, P., Törmä, S., Beetz, J., Weise, M., Liebich, T., 2015. Linked data in architecture and construction.

[11] Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, R., De Meyer, R., Van de Walle, R., Van Campenhout, J., 2011. A semantic rule checking environment for building performance checking, Autom. Constr. 20 (2011) 506-518.

[12] Pauwels, P., Zhang, S., Lee, Y.-C., 2017. Semantic web technologies in AEC industry: a literature overview, Autom. Constr. 73 (2017) 145–165.

[13] Rasmussen, M., Pauwels, P., Lefrançois, M., Schneider, G.F., Hviid, C., Karlshøj, J., 2017. Recent changes in the building topology ontology, in: LDAC2017-5th Linked Data in Architecture and Construction Workshop.

[14] Segaran, T., Evans, C., Taylor, J., 2009. Programming the Semantic Web. O'Reilly Media. (ECPPM), CRC Press. 2014, pp. 135–140.