2019 EC³
2019 European Conference on Computing in Construction
Chania, Crete, Greece
July 10-12, 2019

INCLUDING WIDESPREAD GEOMETRY FORMATS IN SEMANTIC GRAPHS USING RDF LITERALS

Mathias Bonduel[1], Anna Wagner[2], Pieter Pauwels[3], Maarten Vergauwen[1] and Ralf Klein[1]
[1]Department of Civil Engineering, Technology Cluster Construction, KU Leuven, Ghent, Belgium
[2]Department of Civil and Environmental Engineering Sciences, TU Darmstadt, Darmstadt, Germany
[3]Department of Architecture and Urban Planning, Ghent University, Ghent, Belgium

Abstract

The exchange of building data involves both geometric and non-geometric data. A promising Linked Data approach is to embed data from existing geometry formats inside Resource Description Framework (RDF) literals. Based on a study of relevant specifications and related work, this toolset-independent approach was found suitable for the exchange of geometric construction data. To implement the approach in practice, the File Ontology for Geometry formats (FOG) and accompanying modelling method is developed. In a proof-of-concept web application that uses FOG, is demonstrated how geometry descriptions of different existing formats are automatically recognised and parsed.

Introduction

Researchers in both academia and industry are actively investigating the benefits of web technologies to improve the exchange of structured building data, including geometric data (Pauwels et al., 2017). The Semantic Web technology stack is standardised by the World Wide Web Consortium (W3C) and uses Resource Description Framework (RDF) triples as an elementary building block to create graphs of linkable entities. Different methods to include RDF-based geometry descriptions in Linked Data graphs already exist. They have their own dedicated vocabularies, e.g. the OntoBREP[1] or GEOM[2] ontologies, and related toolsets (Perzylo et al., 2015).

Creating geometry representations in RDF according to the approach mentioned above can be achieved by using a dedicated Linked Data geometry modelling tool. Alternatively, geometry coming from regular CAD applications can be converted into RDF-based geometry descriptions. When reusing such geometry descriptions stored as an RDF graph, the geometric data has to be transformed into a readable format for geometry kernels of typical non-RDF applications used in practice. The construction industry, however, also demands a more straightforward method – not related to specific RDF-based geometry toolsets – to easily transfer geometric data. Such data can flow bidirectionally between semantic graphs and the stakeholders' existing geometry processing tools, ideally with as little conversions as possible to minimise conversion errors.

Consequently, methods to introduce already existing and widely used geometry formats (e.g. STEP, OBJ, DWG, etc.) in RDF graphs have to be considered. Two possible alternatives for RDF-based geometry emerge: linking RDF entities of building elements to (1) external geometry files by storing their file location references in RDF literals or (2) RDF literals containing the entire content of such geometry files. This paper focuses primarily on embedding geometry descriptions in RDF literals, but also considers RDF literals referencing external geometry files.

The remainder of this paper contains four sections. The section 'RDF literals and geometric content' covers an analysis of the relevant W3C specifications, existing related implementations and practical requirements to enable the adoption of RDF literals for geometric data. The final part of this first section discusses the feasibility of using RDF literals to store geometry descriptions. Shortcomings of the existing implementations for RDF literals and geometry are addressed in the following section. A new ontology is proposed and validated together with an accompanying modelling method. Section three, 'Proof of concept application', demonstrates how the above can be implemented in applications, while the final section contains the conclusion and addresses future work.

All Uniform Resource Identifier (URI) prefixes mentioned in the remaining of this paper are assembled in Listing 1.

Listing 1: Used URI prefixes in this paper

```
@prefix rdf:
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:
    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix bot: <https://w3id.org/bot#> .
@prefix strdf: <http://strdf.di.uoa.gr/ontology#> .
@prefix geosparql:
    <http://www.opengis.net/ont/geosparql#> .
```

[1]https://github.com/OntoBREP/ontobrep
[2]https://github.com/w3c-geom-cg/geom

DOI:10.35490/EC3.2019.166

```
@prefix geom: <http://bimsparql.org/geometry#> .
@prefix cbim:
    <http://www.coinsweb.nl/cbim-2.0.RDF#> .
@prefix omg: <https://w3id.org/omg#> .
@prefix fog: <https://w3id.org/fog#> .
# namespace for example node instances
@prefix inst: <https://example.org/data#> .
# namespace for a fictive example ontology
@prefix ex: <https://example.org/onto#> .
```

RDF literals and geometric content

In this section, the potential of RDF literals for representing and exchanging geometric content is studied. The first subsection introduces how RDF literals are defined in the relevant web standards, while the second part discusses how other researchers and organisations apply RDF literals for geometric content. Based on current practice in the construction industry, five practical requirements for the use and exchange of geometric data are listed in the third subsection. Finally, the feasibility of RDF literals to store geometry descriptions is discussed.

The nature of RDF literals

The RDF 1.1 W3C specification[3] determines how RDF literals should be used in Linked Data graphs and refers to other related W3C standards.

While each RDF triple consists of a subject, predicate and object, an RDF literal can only be used in the object position of a triple. Thus, a literal can have an incoming relation but no outgoing links. The three essential parts of an RDF literal are (1) the lexical form, (2) the datatype URI and (3) an optional language tag. The lexical form has to be a single collection of UNICODE characters that corresponds to a certain literal value (value space). The datatype URI refers to a datatype that in turn defines what content is valid (the lexical space), the value space and the mapping between the lexical space and the value space. Each literal has exactly one datatype.

Predefined datatype URIs in the W3C specifications have a fixed referent. The Web Ontology Language or OWL 2 specification[4] for example contains the fixed datatypes owl:real, owl:rational, rdfs:Literal, rdf:PlainLiteral, rdf:XMLLiteral and a selection of XSD datatypes (XML Schema Definition Language) documented in the W3C XSD 1.1 specification Part 2[5] (e.g. xsd:integer, xsd:string, etc.). Additionally, custom datatypes can be defined in an ontology by extending a fixed datatype or creating separate new datatypes.

Datatypes cannot be formally defined as a 'subdatatype' from another datatype, as each RDF literal has exactly one datatype assigned. This distinguishes them from classes and properties in RDF, which can have respectively a 'subclass' or a 'subproperty'.

[3]https://www.w3.org/TR/rdf11-concepts/
[4]https://www.w3.org/TR/2012/REC-owl2-syntax-20121211
[5]https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/

Related work

The application of RDF literals for storing building geometry descriptions or references to geometry files is not new, but research documented online and in literature focuses on rather specific cases or domains. The shortcomings of six existing ontologies are analysed in this subsection. First, two frequently used geospatial ontologies are discussed, followed by three ontologies from the construction domain. Lastly, a more generic and flexible ontology, recently published by the authors of this article, is proposed.

GeoSPARQL

Two geospatial ontologies that allow to store existing geometry descriptions in RDF literals are discussed: the GeoSPARQL and the stRDF ontology.

The OGC GeoSPARQL specification (Open Geospatial Consortium, 2012) describes an ontology with the same name, and a SPARQL extension for 2D spatial querying. The ontology provides terminology to connect any object via an intermediate node to a Well Known Text (WKT) or Geography Markup Language (GML) geometry description embedded in an RDF literal. The datatype property between the intermediate node and the RDF literal is specific regarding the geometry format (geosparql:asWKT or geosparql:asGML). Additionally, the literal has a custom datatype assigned (geosparql:wktLiteral, resp. geosparql:gmlLiteral). While each GML file can reference a standardised Coordination Reference System (CRS) by design, the GeoSPARQL specification allows to mention the CRS inside the lexical form of the literal in case of WKT geometry description.

stRDF

The stRDF ontology was developed together with a SPARQL extension named stSPARQL to allow 2D spatiotemporal querying (Koubarakis et al., 2012). Similar as GeoSPARQL, stRDF allows to link any object via an intermediate node to a WKT or GML geometry description stored in an RDF literal. In contrast to GeoSPARQL, only the (custom) datatypes are specific: strdf:WKT or strdf:GML. The stRDF / stSPARQL implementation also allows to include the used CRS inside the lexical form of the literal in case of WKT.

BimSPARQL

In Zhang et al. (2017), the BimSPARQL extension for SPARQL was designed to spatially query and analyse 3D WKT geometry descriptions connected to building elements from an IfcOWL-based RDF graph. Additional terminology is provided in the geometry module of the sup-

porting BimSPARQL ontologies. Applying this module, the WKT geometry descriptions are stored in an RDF literal without a specific custom datatype but as an xsd:string. The datatype property between the intermediate node and this literal is also generic (geom:asBody).

Building Topology Ontology (BOT)

The Building Topology Ontology (BOT) is developed within the W3C Linked Building Data (LBD) group as a central ontology that can be extended with other modular ontologies (Rasmussen et al., 2017). The current version of BOT (v0.3.0) (W3C Linked Building Data Community Group, 2019) allows to connect any bot:Element or bot:Zone instance directly to a geometry description stored in an RDF literal using the generic datatype property bot:hasSimple3DModel. The BOT ontology does not define format specific datatypes or datatype properties for geometry descriptions stored in RDF literals. Besides the above datatype property, BOT also contains the object property bot:has3DModel to link building elements or zones to RDF-based geometry descriptions or an external geometry files. The property names and definitions suggests that they should only be used to connect to 3D geometry descriptions, thus ignoring 2D geometry.

COINS and ICDD

The COINS project (BIM-Loket, 2016) provides a methodology and vocabulary to annotate and link any group of construction related files using Linked Data technology. The result is called a COINS container, consisting of a number of files and an RDF annotation of these files and referenced files not included in the container. In the COINS approach, the separate files are linked to the RDF annotation graph using RDF literals containing the location of the file combined with the fixed datatype xsd:anyURI. This literal is connected with a generic property to an intermediate cbim:UriProperty instance node, which is again connected to a cbim:DocumentReference instance node. In a similar way, users can link any string literal to a cbim:StringProperty instance node that is connected to the cbim:DocumentReference instance node, to add information about the document type and the document Media type (MIME type) of the annotated file. This method allows users to define custom text descriptions at will, making it hard for software developers to query for specific files as several slightly different descriptions of the same file format might exist. Major parts of the COINS project are incorporated in the Information Container for Data Drop (ICDD) which is now under review for standardisation as ISO 21597.

Ontology for Managing Geometry (OMG)

The recently published Ontology for Managing Geometry (OMG) (Wagner et al., 2019) by the authors of this paper provides a datatype property named omg:hasSimpleGeometryDescription to link to a geometry description (2D or 3D) of any geometry format stored in an RDF literal. Similar to bot:hasSimple3DModel, this relation is not specific, but its scope is broadened to include both 2D and 3D geometry descriptions. OMG does not define any specific datatype properties and custom datatypes, as its scope is to arrange the management of geometry in general, independent of the actual geometry formats used. Inspired by the Ontology for Property Management (OPM) (Rasmussen et al., 2018), the OMG can be used to create level 1, 2 or 3 relations between a building element and its geometry descriptions (Wagner et al., 2019). This gives users the flexibility to use a more complex level 3 pattern (three relations between the building object and the geometry description) that allows version control, or a very simple level 1 pattern (direct link between an element and its geometry description) for easy querying.

Practical requirements for the use and exchange of geometry

Exchanged building geometry can be 2D or 3D, it can be a point cloud of an existing construction recorded by a laser scanner, a detailed geometry of a building product or a conceptual design of a new construction. Looking at current practice in the construction industry, it can be concluded that a wide variety of software and related geometry formats are used in different projects, during different lifecycle phases and by different actors (Pauwels et al., 2011). The following minimal practical requirements for the use and exchange of geometry descriptions using RDF literals are derived from this everyday reality:

1. Geometry descriptions stored in RDF literals can be of any existing geometry format, including text and binary files from open and proprietary formats. The RDF literals can contain 2D and/or 3D geometry descriptions. The content of text encoded geometry files can contain single and double quotation marks and is typically read line by line.

2. The actual geometry file format of the content stored in the RDF literal should be defined explicitly. As a result, software applications can query for the used geometry file formats, without having to analyse the content of each individual literal. Doing so not only helps applications to deal with the wide variety of geometry formats that can be available in an RDF graph, but also prevents the exchange of geometric data that is not supported by a certain application.

3. Some geometry schemas have multiple versions while others have multiple closely related schemas. The COLLADA format for example has a frequently used version 1.4.1, besides a newer version 1.5.0. The ASCII format Well Known Text (WKT) has a binary counterpart named Well Known Binary (WKB) that implements a similar internal data structure. Relations between both geometry schema versions and equivalent schemas should be recorded to easily retrieve geometry descriptions in similar schemas.

4. Applications should be able to unambiguously determine which RDF literals have to be treated together. Some geometry formats such as OBJ and glTF allow additional referenced files. For example the OBJ format defines a .obj file that can reference an additional .mtl file containing information about the materials and textures. Simultaneously, each building object can have multiple geometry representations from different points in time, coming from different actors and in different geometry formats.

5. It should be possible to add relevant metadata about each individual geometry description of a building object. Example metadata properties are file size, author, software that created the geometry descriptions, georeferences, the used scale / units and the up-axis. The last three properties are very relevant if the software application of a stakeholder has to use geometry from different geometry formats coming from other geometry modelling applications.

Feasibility of embedding geometry descriptions in RDF literals

The following subsection discusses the feasibility of using RDF literals to embed geometry descriptions of construction elements, according to the practical requirements and the relevant W3C specifications. Each paragraph discusses if and how each requirement can be fulfilled.

Requirement 1: allow every type of geometry format

The content of binary geometry files can be stored in RDF literals by encoding them in UNICODE characters. The base64 encoding is recommended, as its usage is widespread and encoders / decoders are available in almost every programming language. With this methodology the content of any file (text / binary and open / closed formats) can be embedded in an RDF graph. In the case of text encoded geometry files, text lines are important and the text can contain quotes. This means that each new line and quotation sign should be escaped correctly, depending on the RDF serialisation. With the OMG terminology, both two or three dimensional geometry descriptions can be included.

Requirement 2: explicit geometry formats

A natural way to denote the used geometry formats, is to define custom datatypes in an ontology as has been done in GeoSPARQL and stSPARQL for both WKT and GML literals. Other options are the definition of specific classes in combination with additional intermediate nodes or the definition of specific datatype properties. In order to be able to query for any geometry description stored in RDF literals, the generic OMG datatype property omg:hasSimpleGeometryDescription could be used to link to an RDF literal containing a geometry description.

Requirement 3: link related geometry schemas

Relations between different versions of existing geometry schemas can be established via subproperties or subclasses. However, as established earlier, it is not possible to create subdatatypes. The advantage of subproperties and subclasses is that this information can be used during a reasoning process to infer the superproperties, respectively superclasses. Alternatively, this information can also be included as metadata of the respective classes, properties or datatypes inside the ontology.

Requirement 4: bundling associated files

When a building component is connected to multiple geometry descriptions of which some have one or more associated files, at least one intermediate node between the building element and each geometry description is necessary to know which files are related to the same geometry description. The earlier mentioned OMG ontology provides vocabulary to do exactly this, which would in this case result in a level 2 (one intermediate node) or level 3 (two intermediate nodes) geometry pattern. As the referenced files (e.g. one or even multiple material files (.mtl)) are listed in the main file (e.g. the .obj file) via their file names, these names have to be stored in the graph as well.

Requirement 5: metadata for individual geometry descriptions

Relevant metadata related to the individual geometry descriptions can be added only when using at least one intermediate node between the building component and the RDF literal. The vocabulary for these metadata properties has to be documented in an ontology.

Analysis result

RDF literals – as defined in the current W3C specifications – can store the content of every kind of geometry format, making it possible to embed any geometry description directly in an RDF graph. As discussed in the related work subsection, most existing ontologies for describing such RDF literals either focus on a limited amount of specific geometry formats (e.g. GeoSPARQL, stRDF / stSPARQL

and BimSPARQL) or exclude 2D geometry descriptions (BOT). OMG on the other hand makes it possible to link to any geometry description stored in an RDF literal and is not geometry format specific. To make the approach more practical in real world situations, the following three issues have to be addressed. First, there should be a method for applications to unambiguously distinguish between RDF literals referencing external geometry files, literals embedding text-based and encoded binary geometry descriptions. Secondly, the specific geometry schema of each geometry description has to be made explicit and they should be related to other geometry schemas. Finally, content of associated files has to be connected properly to the content of its main file, including the file names as referenced in the main file. In the following section, these three issues are addressed by the new File Ontology for Geometry formats (FOG) and its related modelling patterns.

FOG: a supporting ontology for geometry descriptions in RDF graphs

This section discusses the design of the File Ontology for Geometry formats (FOG) and the accompanying Linked Data modelling patterns. FOG is designed as an OWL ontology that can be used together with the Ontology for Managing Geometry (OMG). Therefore, defining generic properties for linking building components to geometry descriptions is out of scope for FOG, as it is already covered by OMG. Additionally, terminology for properties to add metadata to individual geometry descriptions is not included in FOG. The design decisions for the FOG ontology and related ABox modelling conventions are explained via three Design Questions (DQ) derived from the feasibility analysis in the previous section.

DQ1: How to distinguish between the different applications of RDF literals?

Both text and encoded binary geometry descriptions can be embedded in an RDF literal, while a literal can also store a reference to an external geometry file.

First, the content of a literal can be a URL referring to an external geometry file that can be downloaded from a weblocation or is stored locally. A parser that recognises this, will know that it should look for an external file. Secondly, if a parser can recognise that the geometry description is embedded in an RDF literal and is text-based, it can be configured to unescape newline and quotation signs. Finally, if an encoded binary geometry description is embedded in an RDF literal, the type of encoding should be known – ideally without having to access the lexical form of the literal – so that the right decoder can be called by a parser. The type of encoding (e.g. base64, base122, hexadecimal, etc.) is unrelated to the binary geometry format used.

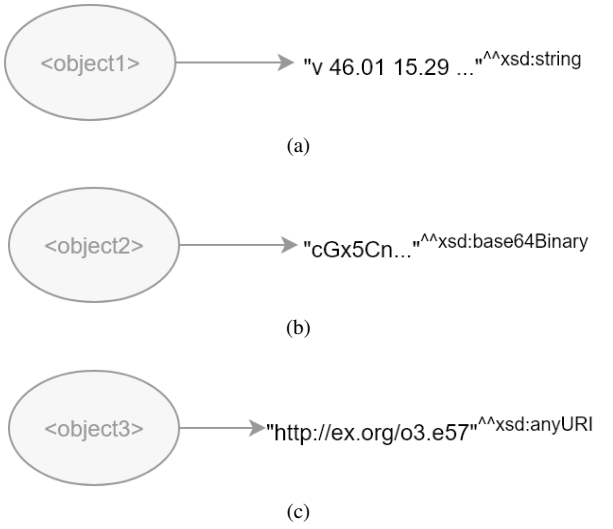

Figure 1: Datatypes used to distinguish between geometry descriptions embedded in RDF literals (a,b) (incl. encoding), and external geometry files referenced in an RDF literal (c).

Software applications can be prepared to deal with the above situations, as long as this information can be retrieved for each geometry description. As a result, stakeholders do not need to manually interpret this kind of information every time geometry is exchanged.

In the proposed modelling pattern, the datatype of each literal is used to add information about the encoding of its lexical form as depicted in Figure 1(a) and 1(b). The fixed datatype xsd:string can be used for a text-based geometry description while the fixed datatypes xsd:base64Binary and xsd:hexBinary can be used for encoded binary data. Alternatively, it is also possible for developers to define other binary encodings as custom datatypes in an ontology. Finally, datatypes can also be used to distinguish between embedded geometry descriptions (text or binary encoded), and references to geometry files outside the RDF graph (Figure 1(c)). In this case, the fixed datatype xsd:anyURI can be used when the lexical form of the literal contains the location of a file, similarly as has been proposed in COINS. Accordingly, no vocabulary has to be introduced in FOG to address this first Design Question as the fixed datatypes suffice in most cases.

DQ2: How can the used geometry format of each RDF literal be defined?

Stakeholders should be able to focus more on the content of the exchanged geometry instead of the formats. If the geometry format of a geometry description can be uniquely identified with FOG terminology, applications can unambiguously request geometry in their supported or

even preferred formats. Additionally, it should be possible for applications to retrieve the geometry format of each geometry description in an RDF graph, without having to access the lexical form of each such RDF literal.

As a consequence, FOG has to provide the necessary terminology to assert the used geometry format of each geometry description. This type of information can be modelled in several ways using custom datatypes, specific classes or specific datatype properties.

Custom datatypes

An initial method could be the definition of custom datatypes in FOG (see Figure 2), similarly as has been done in stRDF / stSPARQL and GeoSPARQL. However, as stated in the W3C specifications, an RDF literal has exactly one datatype and in DQ1 it was already proposed to use the datatype to store other information. Additionally, it is not possible to create a hierarchy of custom datatypes that can be used by reasoners, as there is no such thing as a 'subdatatype'.

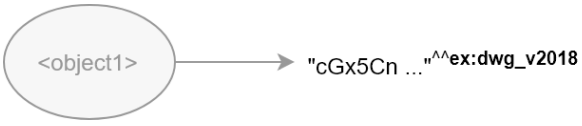

Figure 2: First method: custom datatypes

Specific classes

The class of an additional intermediate node could denote the used geometry format in the content of a connected RDF literal. By defining subclasses, reasoners can be used to infer the class hierarchy of related geometry formats (see Figure 3). However, to group RDF literals that have to be treated together (see feasibility analysis of Req. 4), another additional node is needed between the building element and this node indicating the geometry format. As a consequence, this method tends to result in a relative verbose graph structure.

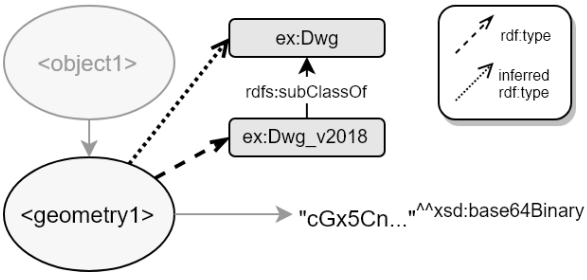

Figure 3: Second method: specific classes

Specific datatype properties

Alternatively, specific datatype properties that point to the RDF literal with the geometry description can be used (Figure 4). This method allows to use the datatypes for the purposes defined in DQ1 and at the same time, the graph is less verbose than the method involving specific classes. OWL allows to define subproperties, similarly as subclasses, that can be used during a reasoning process. Based on the above reasons, the method with specific datatype properties was selected to fulfil this Design Question.

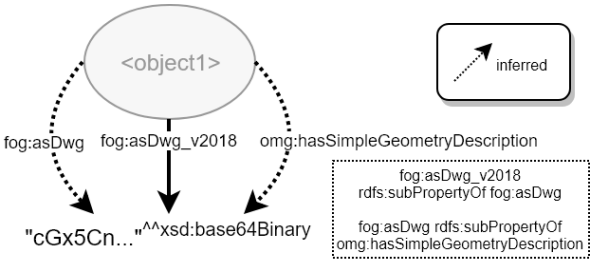

Figure 4: Third method: specific datatype properties

The FOG ontology is thus populated with a taxonomy of datatype properties that refer to existing geometry formats. FOG extends OMG, as the root datatype property is omg:hasSimpleGeometryDescription. The taxonomy then defines FOG subproperties per geometry format (e.g. fog:asDwg), which splits per version of the geometry format (e.g. fog:asDwg_v2018). Some geometry formats can also consist of different (related or unrelated) files, so an additional set of subproperties is defined in those cases (e.g. fog:asGltf_v2.0-gltf and fog:asObj_v2.0-glb).

FOG does not define rdfs:range restrictions on the datatype properties as the same property can be used to link to RDF literals with different datatypes depending on the situation. For example the property fog:asGltf_v2.0-glb can be used to link to a literal with a xsd:base64Binary, xsd:hexBinary, xsd:anyURI or even a custom datatype for other binary-to-text encodings.

Besides the FOG datatype properties, used to link to geometry descriptions stored or referenced in RDF literals, FOG also defines specific object properties (e.g. fog:asGeomOntology) as subproperties of omg:hasComplexGeometryDescription to link to RDF-based geometry descriptions that use dedicated ontologies such as GEOM, OntoBREP and OntoSTEP.

DQ3: How can associated RDF literals be grouped?

If a building object has multiple geometry descriptions of which at least one has associated files, an additional intermediate node between the building component and each

geometry description is necessary. The OMG ontology provides the necessary terminology to do so and can be used together with FOG as shown in Figure 5 for a level 2 geometry pattern (one intermediate node). The content of associated files and their original file names have to be stored in an unambiguous manner along the main geometry description. Each omg:Geometry instance node is connected via the fog:hasReferencedContent property to a corresponding fog:ReferencedContent instance node, if applicable. This node then connects to the file name via an rdfs:label and the actual content of the file via the applicable FOG datatype property as defined in DQ2. When geometry descriptions are not embedded in the graph, but links to external geometry files are defined instead, the correct referenced file name of associated files can be available in the reference string itself. An advantage of using OMG level 2 is the option to add metadata related to individual geometry descriptions as demonstrated in Figure 6. Terminology to define metadata has to be provided by an existing or a new ontology, but is covered by neither FOG nor OMG.

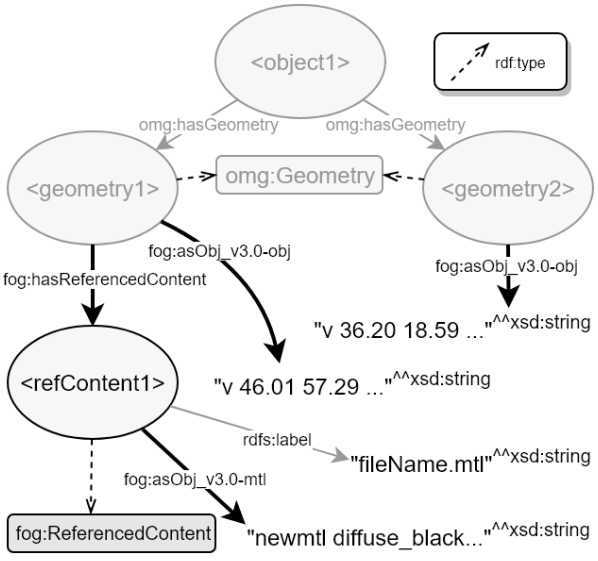

Figure 5: Grouping RDF literals for geometry formats with multiple files

Summary

The proposed modelling method together with the designed FOG ontology, allow users to define specific relations between building components and geometry descriptions. This makes it possible to query for geometry descriptions of specific geometry schemas, or all geometry descriptions independent of the geometry schema. The defined datatype properties from FOG can be used to connect to RDF literals containing either an embedded

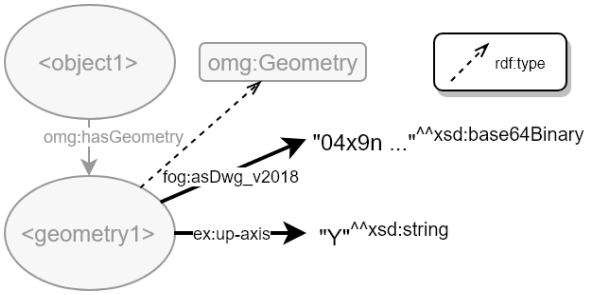

Figure 6: Adding metadata to individual geometry descriptions

geometry description or a reference to an external geometry file. Similarly, object properties are defined in FOG to link to RDF-based geometry descriptions using dedicated geometry ontologies such as GEOM, OntoBREP, etc. The resulting ontology is thus flexible enough to be used in a wide variety of cases involving geometry descriptions in Linked Data.

Metadata about each geometry schema is added to the property definition in the ontology. This includes – if applicable – the file extension, links to associated geometry schemas defined in FOG (dependencies), online specifications, IANA Media (MIME) types and links to related entries in DBpedia and / or Wikidata.

It is expected that the current version of the ontology will be extended over time to include missing and new geometry schemas, and this should be a community effort. As the ontology is published on GitHub[6], users can easily propose new subproperties for the existing ontology. Besides the raw ontology on GitHub, a human readable HTML documentation page is provided via a HTTP redirect from its base URI[7].

Proof of concept application

A proof of concept web application was implemented to visualise geometry stored in an RDF graph. The application, conceptually demonstrated in Figure 7, communicates via SPARQL queries with an RDF triplestore that contains the FOG ontology (TBox) and ABox RDF triples following the modelling principles defined in the previous section. The triplestore answers the query with a JSON response containing at least the three variables: (1) `value` (the lexical form of the literal), (2) `datatype` and (3) `property` (referring to the geometry schema). The decoder module of the application uses the returned `datatype` of each result to know if and how it should decode the returned `value` variable, or if it should load an external geometry file. With a switch operator each `property` variable is evalu-

[6]https://github.com/mathib/fog-ontology
[7]https://w3id.org/fog#

ated and the correct three.js geometry loader – if available – is called to visualise the geometry description in the 3D web viewer. This last module uses the open source three.js library to create a WebGL-based visualisation. The geometry is coloured differently per geometry schema as indicated in a user interface legend. Additionally, the switch also creates and activates a download-to-file button for each geometry description, including the ones that cannot be visualised by three.js.

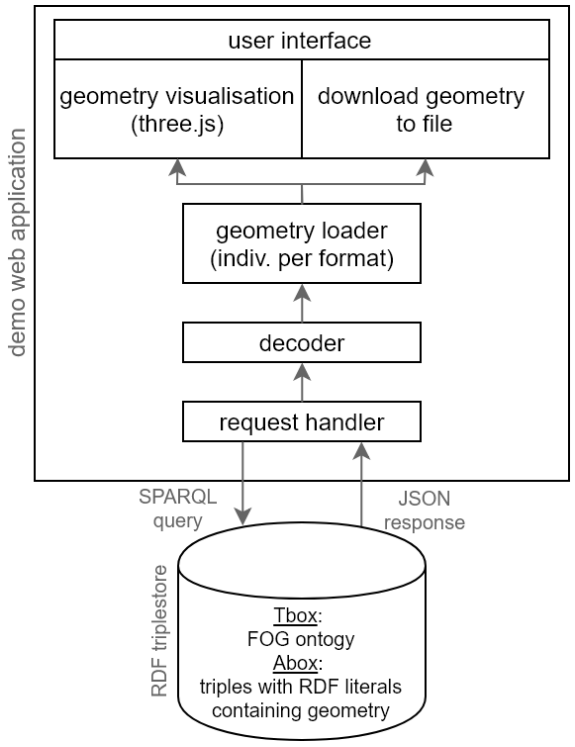

Figure 7: Conceptual diagram of the demo web application

Two sample ABox datasets are published in the GitHub repository of the FOG ontology: the first only contains the graph structure and dummy content for the RDF literals, while the second contains the same graph structure with actual geometry descriptions. The first dataset is used in the publicly available sparql-visualizer demo[8] to demonstrate how it can be queried, while the second dataset is used by the demo web application. See Figure 8 for an overview of the graph structure of the first dataset. The sample datasets contain two columns (instances of bot:Element) consisting of three aggregated parts (capital, shaft and base, also instances of bot:Element). Both the columns and the aggregated parts have been connected via omg:hasGeometry to at least one intermediate omg:Geometry instance node each. Each intermediate node is then connected to one geometry description using the specific properties of FOG, each related to a specific geometry schema. The second sample graph contains embedded geometry descriptions (.obj / .mtl, .glTF, .step, binary .ply and .glb), RDF-based geometry descriptions (GEOM ontology) and linked external geometry files (ASCII .ply, .dae and .e57) stored on Github. The RDF triplestore also contains the FOG ontology (TBox), to allow enhanced querying. The proof of concept application sends the query from Listing 2 to a connected RDF triplestore to get all geometry descriptions available in the default graph. To keep the query clear in this example, it ignores associated files. This query does not need reasoning as it queries the FOG ontology directly, which it assumes to be available in the named graph <http://ontologies.org/fog/>. Figure 9 shows the visualised geometry and the geometry descriptions that can be downloaded in the web application. Geometry descriptions using geometry schemas such as STEP, the RDF-based GEOM ontology and the E57 format, cannot be visualised by three.js as there is no loader available.

[8]https://madsholten.github.io/sparql-visualizer/?file=https://raw.githubusercontent.com/mathib/fog-ontology/master/examples/fog-demo.json

Listing 2: SPARQL query used by the demo web application

```
SELECT ?value ?property ?datatype WHERE {
    ?geometry ?property ?value ;
        a omg:Geometry .
    GRAPH <http://ontologies.org/fog/>{
        ?property RDFs:subPropertyOf* ?omgProp .
        FILTER (?omgProp IN
            (omg:hasSimpleGeometryDescription ,
            omg:hasComplexGeometryDescription)) .
    }
    BIND(DATATYPE(?value) AS ?datatype)
}
```

Conclusion

A list of five minimal, practical requirements for the use and exchange of geometry descriptions within RDF graphs was presented. It was concluded – based on the relevant W3C specifications and existing related ontologies – that RDF literals have the potential to fulfil these requirements. However, a clear and uniform modelling method and related ontology was still missing. The presented FOG ontology and related modelling patterns make it possible to embed geometry of any existing format in RDF literals. The geometry descriptions can contain 2D or 3D geometry, and the geometry schema can be open or proprietary. The suggested method can also be used to link to external geometry files and RDF-based geometry descriptions. Where RDF-based geometry descriptions demand specific applications to interpret them, the presented approach allows to reuse existing geometry formats and a wide variety of related toolsets. Thus, it is better suited for the exchange of geometry between a diverse group of stakeholders.

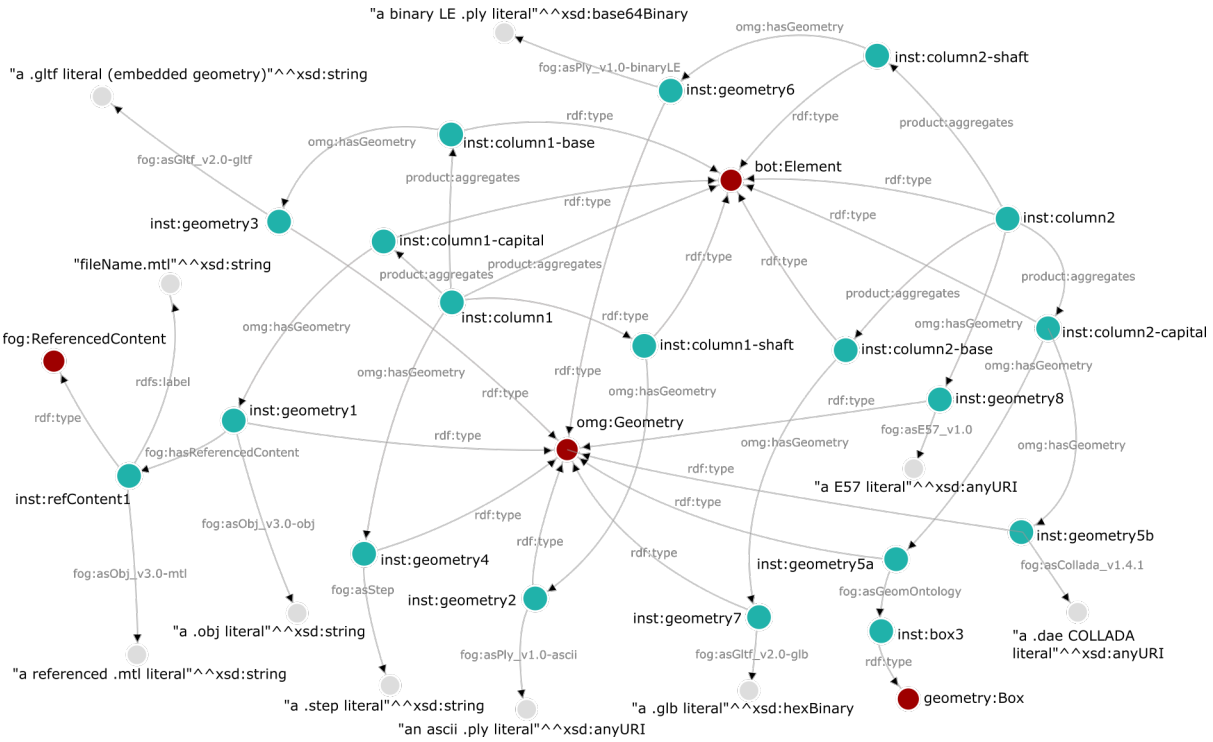

Figure 8: Sample ABox dataset with dummy literals (grey: RDF literals ; red: classes ; green: instance nodes)

The FOG ontology is modular as it extends the more generic OMG ontology to define geometry schema specific relations between building elements and their connected geometry descriptions. This is useful, for example in the case of an existing building, as an individual component can be represented in a 2D as-built drawing, a (part of a) point cloud and a 3D BREP model at the same time. Additionally, multiple geometry schemas can be used simultaneously during the design of new buildings.

OMG and FOG can be used together with other Linked Building Data (LBD) ontologies such as BOT (building topology), PRODUCT (classification of building components) and PROPS / OPM (properties of building elements or zones), depending on the needs of a building project at a certain time.

The proposed FOG ontology and modelling patterns were used in a proof of concept web application and applied when creating a set of sample triples and queries that was shared online using the *sparql-visualizer* tool. The demo application visualises the geometry descriptions coming from a connected RDF triplestore and allows to download any geometry description to a file. The decoder module of the application uses the datatype of each literal to know if and how the content should be decoded, or if an external geometry file has to be loaded. The geometry is automatically loaded and visualised by the correct loader based on the defined FOG properties identifying the used geometry schema.

Before the large scale application of geometry descriptions embedded in RDF literals, the potential influence of literal size on the query performance has to be evaluated. Minimal metadata related to each individual geometry description such as units / scale, up-axis, georeference, etc. is essential if an application needs to deal with geometry descriptions of any geometry schema coming from different modelling applications. Additionally, it is also useful to add metadata related to the modelling accuracy if the geometry description was modelled based on survey data such as point clouds. The approach presented in this paper allows to add such metadata if at least one intermediate node is used between the building object and the geometry description. The terminology to specify geometry-related metadata has yet to be defined in an ontology as existing metadata ontologies such as Dublin Core and PROV-O are rather generic. In contrast to GeoSPARQL and stSPARQL for WKT and GML geometry descriptions and BimSPARQL for WKT geometry descriptions, it is not yet possible to execute spatial queries on a dataset constructed with FOG and the related modelling method. Future research should study how this can be implemented for other, frequently used geometry formats in the construction industry besides the WKT and GML formats. Finally, this approach of embedding geometry descriptions in RDF literals has to be compared in detail with other approaches to

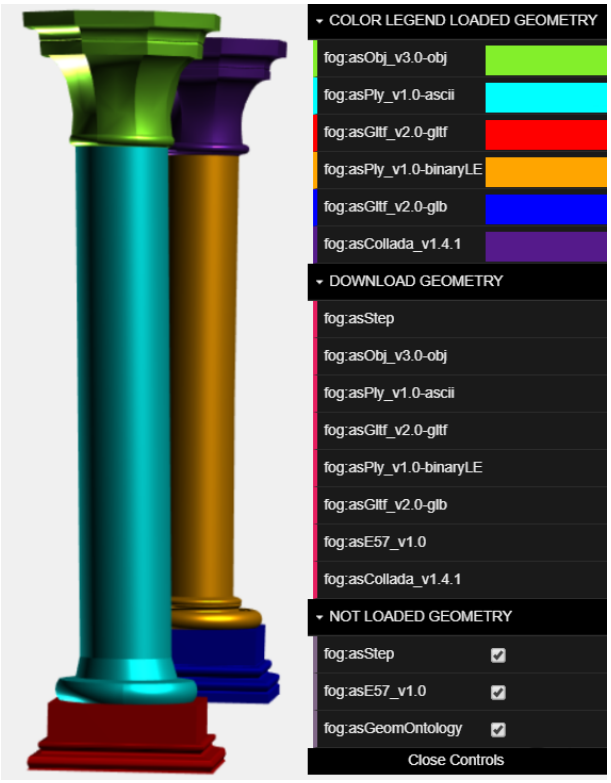

Figure 9: Screenshot of the demo web application

include geometry descriptions in Linked Data graphs such as linking to external geometry files and RDF-based geometries. This will help users to select the right approach for their specific use case.

Acknowledgements

This research is funded by the Research Foundation Flanders (FWO) in the form of a personal Strategic Basic research grant.

References

BIM-Loket (2016). COINS Building Objects Virtual Construction. Accessed March 20, 2019. http://www.coinsweb.nl/.

Koubarakis, M., M. Karpathiotakis, K. Kyzirakos, C. Nikolaou, and M. Sioutis (2012). Data Models and Query Languages for Linked Geospatial Data. Accessed March 20, 2019. http://strabon.di.uoa.gr/files/survey.pdf.

Open Geospatial Consortium (2012). GeoSPARQL - A Geographic Query Language for RDF Data. Accessed March 20, 2019. https://www.opengeospatial.org/standards/geosparql.

Pauwels, P., D. Van Deursen, J. de Roo, T. Van Ackere, R. de Meyer, R. Van de Walle, and J. Van Campenhout (2011). Three-dimensional information exchange over the semantic web for the domain of architecture, engineering, and construction. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 25*(4), 317–332. DOI: 10.1017/S0890060411000199.

Pauwels, P., S. Zhang, and Y.-c. Lee (2017). Semantic web technologies in AEC industry : A literature overview. *Automation in Construction 73*, 145–165. DOI: 10.1016/j.autcon.2016.10.003.

Perzylo, A., N. Somani, M. Rickert, and A. Knoll (2015). An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015)*, pp. 4197–4203. IEEE. DOI: 10.1109/IROS.2015.7353971.

Rasmussen, M. H., M. Lefrançois, M. Bonduel, C. A. Hviid, and J. Karlshøj (2018). OPM: An ontology for describing properties that evolve over time. In *Proceedings of the 6th Linked Data in Architecture and Construction Workshop (LDAC), CEUR Workshop Proceedings*, Volume 2159, London, UK, pp. 23–33.

Rasmussen, M. H., P. Pauwels, C. A. Hviid, and J. Karlshøj (2017). Proposing a Central AEC Ontology That Allows for Domain Specific Extensions. In *Proceedings of the Joint Conference on Computing in Construction (JC3), LC3 2017: Volume I*, Heraklion, Greece, pp. 237–244. DOI: 10.24928/JC3-2017/0153.

W3C Linked Building Data Community Group (2019). Building Topology Ontology. Accessed March 20, 2019. https://github.com/w3c-lbd-cg/bot.

Wagner, A., M. Bonduel, P. Pauwels, and U. Rüppel (2019). Relating geometry descriptions to its derivatives on the web. In *Proceedings of the European Conference on Computing in Construction (EC3 2019)*, Chania, Crete, Greece.

Zhang, C., J. Beetz, and B. de Vries (2017). BimSPARQL: Domain-specific functional SPARQL extensions for querying RDF building data. *Semantic Web Journal 1*, 1–17. DOI: 10.3233/SW-180297.