HAL
open science

From CityGML to OWL

Diego Vinasco-Alvarez, John Samuel Samuel, Sylvie Servigne, Gilles Gesquière

To cite this version:

Diego Vinasco-Alvarez, John Samuel Samuel, Sylvie Servigne, Gilles Gesquière. From CityGML to OWL. [Technical Report] LIRIS UMR 5205. 2020. hal-02948955

HAL Id: hal-02948955

https://hal.science/hal-02948955

Submitted on 25 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

From CityGML to OWL

Diego Vinasco-Alvarez, John Samuel, Sylvie Servigne, Gilles Gesquière

Technical Report Submitted to LIRIS, UMR 5205

TABLE OF CONTENTS

1. Introduction 1

1.1. General Context 1

1.2. Urban Data Within Graphs and Ontologies 2

2. From CityGML Urban Data to Linked Data: A Proposed Approach 3

2.1. Transforming CityGML Instances into RDF 6

2.2. Generating a CityGML Ontology from XML Schema 11

2.3. Integrating GeoSPARQL 14

3. Conclusion 16

4. Acknowledgements 18

5. References 19

1. INTRODUCTION

Urban environments are naturally complex and heterogeneous sources of information. The data they generate, these “geospatial” urban data, can take many forms, descriptions, and may follow many different standard formats. The data presented in these formats are generally incompatible with each other and cannot always be easily queried. This poses a problem for researchers, historians, and city planners – among many other stakeholders – as it becomes difficult to take all the available data into account without conversion or making several, distinct queries in separate databases.

It is critical for these groups to understand what this information represents and how it inter-relates among itself. Observing these relationships and how they evolved over time can give a more detailed view of how the past decisions of city planners impacted cities historically and in the present, which in turn allows more informed decision making about planning and developing cities for the future.

Is there a way to link and query these various forms of urban data side-by-side, and if yes, how can we accomplish this? What do we precisely gain from this representation? And what are the limitations of the format?

1.1. General Context

Geospatial urban data or multi-dimensional urban data, as they contain a locational or spatial context, fall under many types of representations and visualizations. Multi-dimensional urban data are sometimes referred to as n-D or n-dimensional data, as they can be represented in 2D (floorplans, satellite imagery), 3D (CAD models, terrain topology), with thematic and additional information such as administrative documents, sketches, and articles, and temporally as they change over time. In recent years, many different formats have seen a rise in use to model, store, and visualize this broad type of urban information. Spatial Data Infrastructures (SDI) can provide standardized frameworks for structuring and sharing geospatial data [1], Geographic Information Systems (GIS) and Building Information Models (BIM) provide tools and models for the analysis and capture of 3D and geospatial data [2]. The Open Geospatial Consortium (OGC) – an international consortium of more than 500 businesses, government agencies, research organizations, and universities – provides over 60 unique standardized geospatial data formats and frameworks [3]. With such a large volume of methods for serializing urban data, the need to inter-relate this information is ever present to create a more complete representation of the urban landscape.

A possible approach to this problem is to implement semantic web technologies and move multi-dimensional urban data into linked-data models. In order to manipulate an existing body of this data alongside other formats, we propose using a conversion tool to transform the data into a linked data, supported by an ontology, allowing us to take advantage of existing standards and to permit querying this data alongside other linked data. The dataset we propose to convert will be based in CityGML; an XML based SDI extension of GML proposed by the OGC for modeling virtual cities in 3D. Its final representation will be in OWL/RDF; a graph-based ontology language proposed by the World Wide Web Consortium (W3C).

1.2. Urban Data Within Graphs and Ontologies

Resource Description Framework (RDF) is a standard model for linked data interchange information on the Web. Its structure consists of sets of assertions called triples. Triples are composed of subjects, predicates, and objects (fig. 1) where the subject is the topic of the assertion, the predicate describes the relationship between the subject and object, and the object is either another subject or a primitive datum such as a string or integer. With this structure, we can create a graph to describe the relationships between points of information by using the subjects and objects as nodes and the predicates as the edges between the nodes.

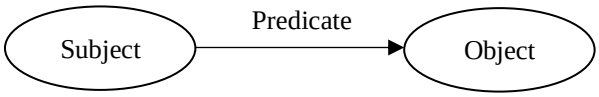

Figure 1. RDF triple structure

The semantic web builds on the framework laid out with RDF. RDF Schema (RDFS) and the Web Ontology Language (OWL) allow us to add context to RDF triples. Categories or types of objects are described with *Classes*, their instances are called *Individuals*, their characteristics are called *Datatypes*, and the relationships between all these concepts are called *Properties* (fig. 2). Although research in implementing this semantic web stack are still ongoing, several research efforts have already been made [1], [2], [4]–[7] in representing geospatial and urban data in linked data and ontological formats. Furthermore, there is an official semantic web resources proposed by the OGC for representing this data: GeoSPARQL – a geospatial ontology and a functional extension to the RDF query language SPARQL. However, none of these approaches provide a semantic representation of CityGML's features nor a method for converting CityGML features into RDF/OWL.

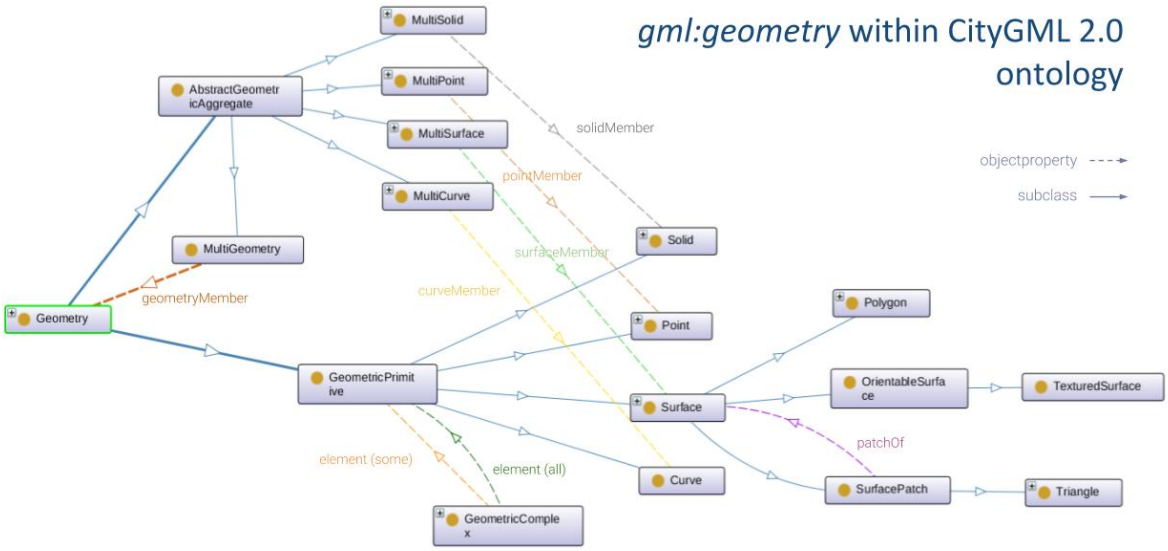

Figure 2. CityGML 2.0 Ontology: Geometry Classes and Properties [8]

The SPARQL Protocol and RDF Query Language (SPARQL) is one of, if not, the most common means of querying RDF data. Generally, queries are SQL-like as they utilize SELECT-WHERE statements to extract data from RDF. Queries are formulated in a triple structure - like RDF - with the intention of returning statements that match the patterns denoted in the query. The GeoSPARQL extension to these queries adds spatial functions, such as *geof:sfIntersects*, and *geof:sfOverlaps*, that return datum which

satisfy the spatial relations described by these functions, giving us a powerful tool for retrieving and analyzing the CityGML data after conversion.

This report will enumerate the work done by exploring the previously mentioned proposal as follows: Section 2 proposes the conception and development of a proof of concept tool[1] to convert CityGML information into linked data and its experimentation during the project. Finally, section 3 will conclude with a synthesis of the results and perspectives gained from the project.

2. From CityGML Urban Data to Linked Data: A Proposed Approach

To create a proof of concept tool for the generation of geospatial data from CityGML, an XSLT-based pipeline was created, specifically based on [8]–[12] and proceeds as follows (fig. 3):

1. The generation of an intermediate XSLT to transform CityGML instances to RDF from the GML and CityGML application schema.
2. Generation of CityGML RDF instances using the previously generated XSLT and CityGML data from the metropole of Lyon.
3. The creation of an OWL ontology to describe the resulting CityGML instances from the GML and CityGML application schema.

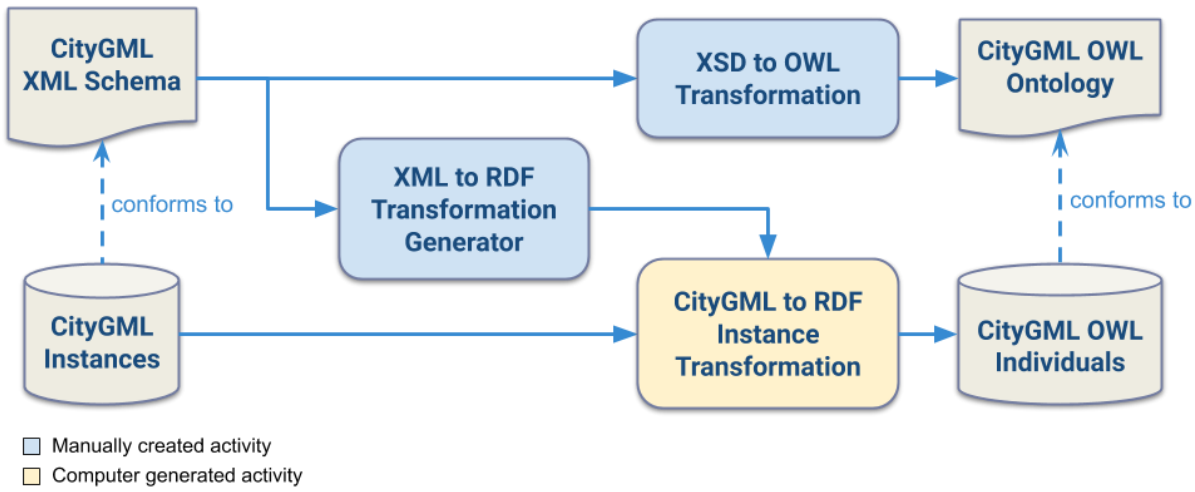

Figure 3. General Approach pipeline

During each transformation, the resulting information must be logically consistent and maintain its interoperability from CityGML. To ensure this, several challenges need to be overcome. For instance, the RDF structure and types generated from the GML instances must conform to the model described by the ontology. How can mappings be created to ensure this? CityGML application schema often implements elements that do not have a direct equivalent in OWL or RDF. How should these elements be represented to best describe CityGML as semantic data? In addition, CityGML schema often draws from elements,

[1] https://github.com/VCityTeam/UD-Graph

types, functionality from external schema such as GML, xLinks and xAL addresses. How should these imports be addressed to preserve their original functionality?

The workflow of the processes in this report is broken up into two pipelines: an instance transformation pipeline and a model transformation pipeline. Both pipelines are treated as activities in a 3rd "complete" pipeline which illustrates the conversion process in its entirety and how the various application schema of GML, CityGML, and other imported schema are combined.

Fig. 4 shows the activity diagram for the instance transformation pipeline. In this workflow, the CityGML schema is passed into an XSLT processor to extract the semantic metadata within and create a second, domain specific, XML to RDF XSLT stylesheet. This second stylesheet can be used to transform XML instance files – that conform to the CityGML application schema – into RDF. The stylesheet implements the OWL vocabulary to produce OWL individuals and contains specialized transformations for GML elements to integrate GeoSPARQL vocabulary. The resulting RDF output file of the second transformation will contain import statements for the CityGML ontology as suggested in [9]. After the instance file is produced, a final "postprocessing" activity is performed to validate the generated RDF. This activity utilizes a Python script with the *lxml* library to parse the RDF graph and primarily removes any duplicate instances or properties. Note that the output documents shown in Fig. 4 are serialized in RDF and use the '.rdf' file suffix, but all outputs still implement the OWL vocabulary whenever necessary.

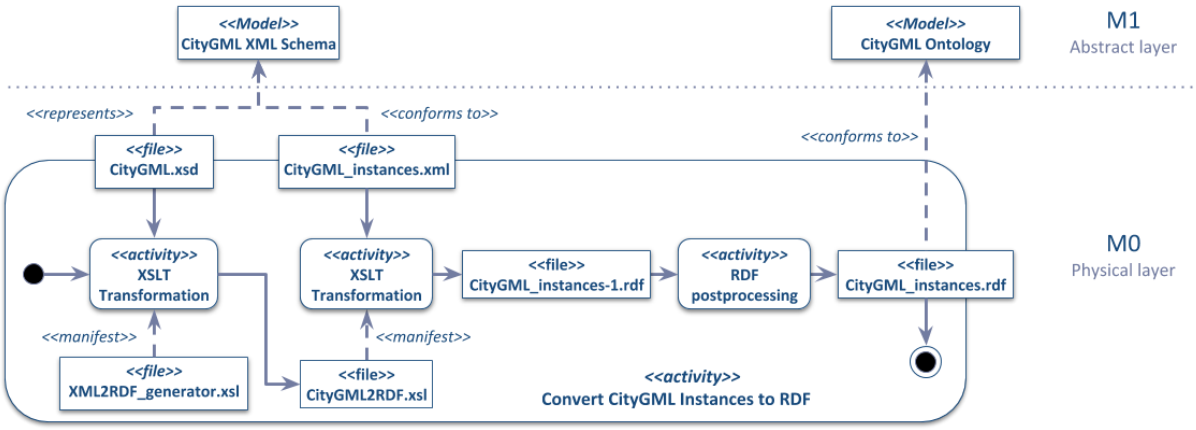

Figure 4. Instance Conversion Activity Diagram

The process for converting the CityGML application schema into an OWL starts with a transformation activity where the XML Schema model to OWL mapping patterns and strategies proposed in [8], [9], [13], [14] are implemented. The mapping patterns used in model and instance transformation are synthesized from these works in XML Schema to OWL transformations. Table 1 denotes the general mapping patterns used to facilitate these transformations. After the transformation activity, a postprocessing script is run on the ontology, like the one utilized in the instance conversion activity. This script imports and integrates the GeoSPARQL ontology, and fully qualifies any namespace prefix used in RDF attributes such as *rdf:resource*. This renders an OWL ontology constrained by the CityGML application schema, thus providing a model for the OWL individuals created in the instance transformation process.

Due to the large number of schemas used to model GML and the modules of CityGML, these processes would have to be run several times per schema document to create a complete CityGML ontology. In addition, the elements in CityGML and GML often refer to and rely on information stored in each other and in external schema documents and thus the XSLT processor must have access to all referenced metadata at the same time. In the XML schema vocabulary, this external information is referenced through *xs:import* and *xs:include* statements that link to the external document. In order to solve this issue for CityGML and GML a "composite" schema made up of all the elements and type declarations of every required schema will be created.

After a composite schema is created, it will be passed to both transformation activities. The schema compilation activity must also normalize or fully qualify the namespace prefixes from each schema document to match. This is required to maintain consistent naming conventions and to simplify namespace resolution during transformations as prefixes may change between schema documents and RDF requires fully qualified namespaces anytime a URI is given. This activity will also remove leading and trailing whitespace between XML elements and from element text to provide the "cleanest" and most compact schema possible. Like the previous activities not powered by an XSLT processor, this XML schema compilation activity is manifested by a Python script using the *lxml* library.

Table 1. Chosen XML Schema to OWL mapping patterns based on [10]–[12], [14] (contributed pattern in orange)

XML Schema Element or Attribute	OWL Target
xs:complexType	owl:Class
xs:simpleType	rdfs:datatype
Global xs:element with a type of an xs:complexType	owl:Class with an rdfs:subClassOf relationship to the type
Global xs:element with a type of an xs:simpleType	rdfs:datatype with an owl:equivalentClass relationship to the type
child xs:element of a xs:complexType with a xs:complexType type	owl:ObjectProperty with an rdfs:domain of the parent type and rdfs:range of its own type
child xs:element of a xs:complexType with a xs:simpleType or native xs datatype type	owl:DatatypeProperty with an rdfs:domain of the parent type and rdfs:range of its own type
xs:attribute	owl:DatatypeProperty with an rdfs:domain of the parent type and rdfs:range of its own type
xs:sequence or xs:all	owl:restriction composed of owl:intersectionOf
xs:group	owl:Class
xs:attributeGroup	owl:Class
substitutionGroup attribute	rdfs:subClassOf
base attribute	rdfs:subClassOf, owl:DatatypeProperty, or owl:ObjectProperty depending on the type of content
minOccurs attribute	owl:minCardinality
maxOccurs attribute	owl:maxCardinality
xs:choice	owl:disjointUnionOf

2.1. Transforming CityGML Instances into RDF

When generating the CityGML instance to RDF transformation, it is important that the patterns created are general enough to be reused and can take advantage of the CityGML vocabulary whenever possible. This process will use the general structure of the XML to RDF mappings proposed in [8], [9] in addition

to some GML to RDF mapping concepts proposed in [1], [11], [15]. There will also be consideration taken so that these mappings work with the ontology to be generated in alongside the RDF data.

In general, three types of mappings will be created from the schema:

1. All global *xs:element* elements that have the type of an *xs:complexType* create a template for generating *owl:NamedIndividuals.*
2. All *xs:complexType*, *xs:simpleType*, *xs:attributeGroup*, and *xs:group* elements create a template that compiles the templates for every possible child element, text, and attribute of the element.
3. All *xs:*attributes or xs*:elements* which are children of *xs:complexTypes* or *xs:groups* create templates for *owl:ObjectProperties* and *owl:DatatypeProperties*.

To illustrate how these mapping types work, we will use an example from the CityGML core module. Fig. 5 shows the schema for the *core:_CityObject* element and its type, *core:AbstractCityObjectType*. The initial transformation of these schema elements, as proposed in this report, would yield a transformation pattern as shown in Fig. 6.

```
<xs:complexType name="AbstractCityObjectType" abstract="true">
  ...
  <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureType">
              <xs:sequence>
                      <xs:element name="creationDate" type="xs:date" minOccurs="0"/>
                      <xs:element name="terminationDate" type="xs:date" minOccurs="0"/>
                      <xs:element name="externalReference" type="ExternalReferenceType"
minOccurs="0" maxOccurs="unbounded"/>
                      <xs:element name="generalizesTo" type="GeneralizationRelationType"
minOccurs="0" maxOccurs="unbounded"/>
                      <xs:element name="relativeToTerrain" type="RelativeToTerrainType"
minOccurs="0"/>
                      <xs:element name="relativeToWater" type="RelativeToWaterType"
minOccurs="0"/>
                      <xs:element ref="_GenericApplicationPropertyOfCityObject"
minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
      </xs:extension>
  </xs:complexContent>
</xs:complexType>
<!-- ============================================================================ -->
<xs:element name="_CityObject" type="AbstractCityObjectType" abstract="true"
substitutionGroup="gml:_Feature"/>
```

Figure 5. core:AbstractCityObjectType and core:_CityObject application schema

To generate this result, three transformations patterns are implemented. First, if a global *xs:element* is found – such as *core:_CityObject* – and it has a type of an *xs:complexType* or has a child *xs:complexType*, a template will be generated to create an individual. The template generated will use the *gml:id* attribute of the element it matches as the *rdf:ID* of the instance as proposed in [11]. If no *gml:id* is available, a unique id will be generated and appended to the local name of the element. Also as suggested in [11] the *rdf:type* of each individual should be generated from its local name and this is well implemented here. However, in this proposal the tertiary conversion to GeoSPARQL will be made based on this *rdf:type* and thus the full name will be used to distinguish between a gml geometry element and a feature. After the individual is named and typed, the template will call a reusable template generated from

the *xs:complexType* type of the element. This template will contain the CityGML to RDF templates to generate the *owl:ObjectProperties* and *owl:DatatypeProperties* for the individual.

```
<xsl:template match="//core:_CityObject">
  <owl:NamedIndividual rdf:about="{if ( @gml:id )
                                   then @gml:id
                                   else concat( local-name(), '_', generate-id() )}">
    <rdf:type rdf:resource="core:_CityObject"/>
    <xsl:call-template name="core:AbstractCityObjectType_Template"/>
  </owl:NamedIndividual>
</xsl:template>

<xsl:template name="core:AbstractCityObjectType_Template">
  <xsl:for-each select="./core:creationDate">
    <xsl:call-template name="core:creationDate_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./core:terminationDate">
    <xsl:call-template name="core:terminationDate_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./core:externalReference">
    <xsl:call-template name="core:externalReference_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./core:generalizesTo">
    <xsl:call-template name="core:generalizesTo_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./core:relativeToTerrain">
    <xsl:call-template name="core:relativeToTerrain_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./core:relativeToWater">
    <xsl:call-template name="core:relativeToWater_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./core:_GenericApplicationPropertyOfCityObject">
    <xsl:call-template name="core:_GenericApplicationPropertyOfCityObject_Property"/>
  </xsl:for-each>
  <xsl:call-template name="gml:AbstractFeatureType_Template"/>
</xsl:template>

<xsl:template name="core:creationDate_Property">
  <core:creationDate>
     <xsl:value-of select="text()"/>
  </core:creationDate>
</xsl:template>

<xsl:template name="core:terminationDate_Property">
  <core:terminationDate>
     <xsl:value-of select="text()"/>
  </core:terminationDate>
</xsl:template>

<xsl:template name="core:externalReference_Property">
  <core:externalReference rdf:resource="{if (./*/@gml:id)
                                         then ./*/@gml:id
                                         else concat( local-name(./*), '_',
                                                      generate-id(./*) )}"/>
</xsl:template>
...
```

Figure 6. Generated CityGML to RDF XSLT from core:_CityObject and core: AbstractCityObjectType

The second type of mapping concerns the transformation of the *xs:complexType* element. It contains 4 sub-transformations that determine the elements and attributes the type could have and create a template that calls additional templates to create the appropriate *owl:ObjectProperty* and *owl:DatatypeProperties*. The sub-transformations are as follows:

1. Collect all descendant *xs:elements* and call the matching templates to generate *owl:ObjectProperty* or *owl:DatatypeProperty* templates depending on the content of the element.

a. If an element belongs to a *substitutionGroup* the corresponding templates for all elements in that group must also be generated.

2. Collect all descendant *xs:attributes* and call the matching templates to generate *owl:DatatypeProperty* templates.

3. If there is an *xs:extension* or *xs:restriction* of any other type, the template for that type must be generated as well according to the *base* attribute.

4. Collect all *xs:group* and *xs:attributeGroup* references and call the matching templates for these groups.

Normally the template for *core:_CityObject* will never be called as it is an abstract *xs:element*, however in the case of the *bldg:Building* element – which is often used in CityGML instances – both elements are in the same *substitutionGroup* and can be used interchangeably*. bldg:Building* also has the complex type *bldg:AbstractBuildingType* which inherits the same complex type template as *core:AbstractCityObjectType*. Because of these relationships, whenever a *bldg:Building* element is declared, its transformation pattern will call the template for *core:AbstractCityObjectType*.

Another XML transformation proposed is the conversion of *xs:complexTypes* with *xs:simpleContent.* Transformation mapping patterns of *xs:simpleContent* and *xs:complexContent* to OWL are proposed in [14] but without a transformations for XML instances of these types. *xs:simpleContent* proved to be one of the more complicated instance transformations to RDF as *xs:complexTypes* are always transformed into *owl:classes* yet *xs:simpleContent* constrains the type to only contain attributes and/or text with no child elements. This implies that these types can sometimes appear as *rdfs:Datatype* elements. For example, Fig. 7 shows the schema for one such type, *gml:MeasureType*. In XML this type could be instantiated as an element with a text value of a double and an attribute of a URI.

```
<xs:complexType name="MeasureType">
  ...
<xs:simpleContent>
  <xs:extension base="xs:double">
    <xs:attribute name="uom" type="xs:anyURI" use="required"/>
  </xs:extension>
</xs:simpleContent>
</xs:complexType>
```

Figure 7. gml:MeasureType application schema

This report bases the proposed instance transformation of these types off of the approach used in [14] with several transformation mappings listed in table 2. In the case that the complex type with a simple content has an *xs:restriction* of a type, a new *rdfs:datatype* and *owl:DatatypeProperty* would be declared in the ontology according to these restrictions and should be used when transforming the instance data into RDF. Note, that when referencing the newly created datatype of an *xs:restriction*, 'Datatype' is appended to the name of the type and the property to avoid overlapping identifiers . Otherwise when the simple content is based on an *xs:extension* of datatype or simple type, there is only an *owl:DatatypeProperty* created that links to the datatype value.

Table 2. XML Schema to RDF intstance mapping patterns

XML Schema pattern	Target instance pattern
<xs:complexType name="TypeName"> <xs:simpleContent> <xs:extension base="nativeDatatypeName">> ... </xs:extension> </xs:simpleContent> </xs:complexType>	<owl:NamedIndividual> <rdf:type rdf:resource="TypeName"/> <hasNativeDatatypeName> someValue </hasNativeDatatypeName> ... </owl:NamedIndividual>
<xs:complexType name="TypeName">> <xs:simpleContent> <xs:extension base="simpleTypeName">> ... </xs:extension> </xs:simpleContent> </xs:complexType>	<owl:NamedIndividual> <rdf:type rdf:resource="TypeName"/> <hasSimpleTypeName> someValue </hasSimpleTypeName> ... </owl:NamedIndividual>
<xs:complexType name="TypeName">> <xs:simpleContent> <xs:restriction base="nativeOrSimpleDatatype">> ... </xs:restriction> </xs:simpleContent> </xs:complexType>	<owl:NamedIndividual> <rdf:type rdf:resource="TypeName"/> <hasTypeNameDatatype> someValue </hasTypeNameDatatype> ... </owl:NamedIndividual>

One departure from the previous XML schema to OWL approaches is how *xs:group* and *xs:attributeGroup* are transformed. In [13], [14] both of these elements are converted into *owl:Classes* as they contain elements and attributes like *xs:complexTypes*. However, in the context of CityGML and XML to RDF data generation these groups serve mostly utility and do not appear in XML instances as individual elements. For instance, Fig. 8 shows a GML group and a complex type which references this group and Fig. 9 shows a CityGML instance that implements them both.

```
<xs:group name="gml:StandardObjectProperties">
  ...
  <xs:sequence>
    <xs:element ref="gml:metaDataProperty" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="gml:description" minOccurs="0"/>
    <xs:element ref="gml:name" minOccurs="0" maxOccurs="unbounded">
      ...
    </xs:element>
  </xs:sequence>
</xs:group>

<xs:complexType name="gml:AbstractGMLType" abstract="true">
  ...
  <xs:sequence>
    <xs:group ref="gml:StandardObjectProperties"/>
  </xs:sequence>
  <xs:attribute ref="gml:id" use="optional"/>
</xs:complexType>
```

Figure 8. gml:StandardObjectProperties and gml:AbstractGMLType application schema

```
<bldg:Building gml:id="A23">
    <gml:name>Example Building</gml:name>
    <gml:description>An example of a building</gml:description>
</bldg:Building>
```

Figure 9. Example CityGML bldg:Building using gml:StandardObjectProperties

In the instance, the *bldg:Building* element of the complex type is clearly instantiating child elements from the *xs:group* however there is no actual reference to the group itself. That is to say, the child elements of *gml:StandardObjectProperties* are instantiated but the group itself is not formally instantiated. Functionally, *xs:groups* and *xs:attributeGroups* serve as a reusable collection elements and attributes. This report argues that because of this behavior, they do not represent *owl:Classes* but are simply features of the XML schema vocabulary. Instead, in XML to RDF transformation, these groups can be transformed into templates that simply contain references to the templates of their properties (fig. 10).

```
<xsl:template name="gml:AbstractGMLType_Template">
  <xsl:if test="@gml:id">
     <xsl:call-template name="gml:id_Property"/>
  </xsl:if>
  <xsl:call-template name="gml:StandardObjectProperties_Template"/>
</xsl:template>

<xsl:template name="gml:StandardObjectProperties_Template">
  <xsl:for-each select="./gml:metaDataProperty">
     <xsl:call-template name="gml:metaDataProperty_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./gml:description">
     <xsl:call-template name="gml:description_Property"/>
  </xsl:for-each>
  <xsl:for-each select="./gml:name">
     <xsl:call-template name="gml:name_Property"/>
  </xsl:for-each>
</xsl:template>
```

Figure 10. Generated CityGML2RDF gml:AbstractGMLType and gml:StandardObjectProperties XSLT Templates

Transformations will also map the original GML literal values to an RDF triple using GeoSPARQL's *geo:asGML* datatype property, if an instance's type is in the substitution group of *gml:_Geometry*. This process is covered in detail in section 2.3. Once an instance document is transformed, it must be scanned for malformed RDF triples and fully qualifies any RDF attributes that contain prefixed URI strings such as *rdf:resources*, *rdf:type*, and *rdf:about*. These transformations follow the "garbage in, garbage out" concept that poorly formed data input into a program, will produce nonsensical results, and thus assume that the GML and CityGML instance documents provided are well structured and conform to their application schema. If this assumption is met, the resulting data should conform to the ontology transformation discussed in the following section.

2.2. Generating a CityGML Ontology from XML Schema

During development, several choices were made to create a transformation of the CityGML application schema that respects its original structure while maintaining logical consistency, even when considering the inferences of an OWL reasoner. In particular: what should be done with schema elements which have no direct representation in OWL? How to standardize and automate namespace and identifier generation? And finally, how should all these things be considered under the context of generating RDF data from CityGML alongside this ontology?

The first of these choices that was made was regarding the transformation of schema elements with no direct representation in OWL. In the case of *xs:choice* a combination of *owl:intersectionOf*, *owl:unionOf*, and *owl:complementOf* are suggested in several approaches [9], [13], [14], [16]. In description logic, unions, intersections, and complements are analogous to logical *'and'*, *'or'*, and *'not'* relationships,

respectively. More precisely, the use of *xs:choice* is analogous to the description of a class that contains the group of *'exclusive or'* (XOR) properties. In the case of two properties '*A*' and '*B*', a class created from an *xs:choice* statement would have either *A* or *B* but not both, as shown in Figure 11 and equation 1.

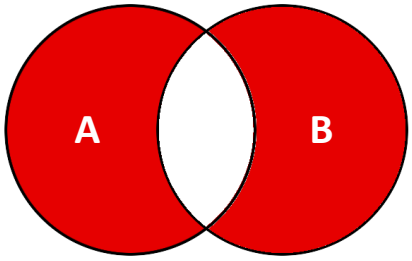

Figure 11. Description of a class with axioms A XOR B

$$(A \cup B) \cap (A \cap B)^C \tag{1}$$

This approach, however, requires an exponentially growing number of statements as more *xs:choice* children are transformed. This is because each intersections of each statement must be declared pairwise disjoint as shown in Figure 12 and equation (2).

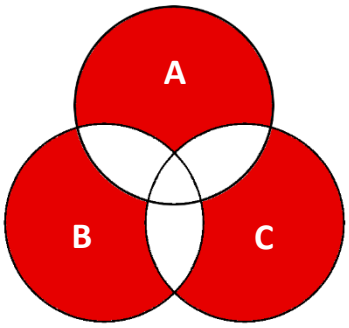

Figure 12. Description of a class with axioms A XOR B XOR C

$$(A \cup B \cup C) \cap ((A \cup B) \cap (B \cup C) \cap (A \cup C))^C \tag{2}$$

While both of these approaches are logically valid for representing CityGML schema in OWL, an approach using OWL-2's *owl:disjointUnionOf* was experimented with (fig. 14), which provides a more concise transformation. The example in Fig. 13 could be converted into the RDF in Fig. 14. This result represents the disjoint union of the class that has either has the property *#name* or *#uri* but not both. This implementation was tested in Protégé[2] with the HermiT[3] reasoner. An individual 'someReference' was

[2] https://protege.stanford.edu/
[3] http://www.hermit-reasoner.com/

instantiated with both *#name* and *#uri* datatype properties. The reasoner returned the following explanation in Fig. 15 and declared the ontology inconsistent, as intended.

```
<xs:complexType name="ExternalObjectReferenceType">
  <xs:choice>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="uri" type="xs:anyURI"/>
  </xs:choice>
</xs:complexType>
```

Figure 13. Example complexType schema

```
<owl:Class rdf:about="#ExternalObjectReferenceType">
   <owl:disjointUnionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#name"/>
         <owl:someValuesFrom rdf:resource="xs:string"/>
      </owl:Restriction>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#uri"/>
         <owl:someValuesFrom rdf:resource="xs:anyURI"/>
      </owl:Restriction>
   </owl:disjointUnionOf>
</owl:Class>
```

Figure 14. xs:choice representation with owl:disjointUnionOf

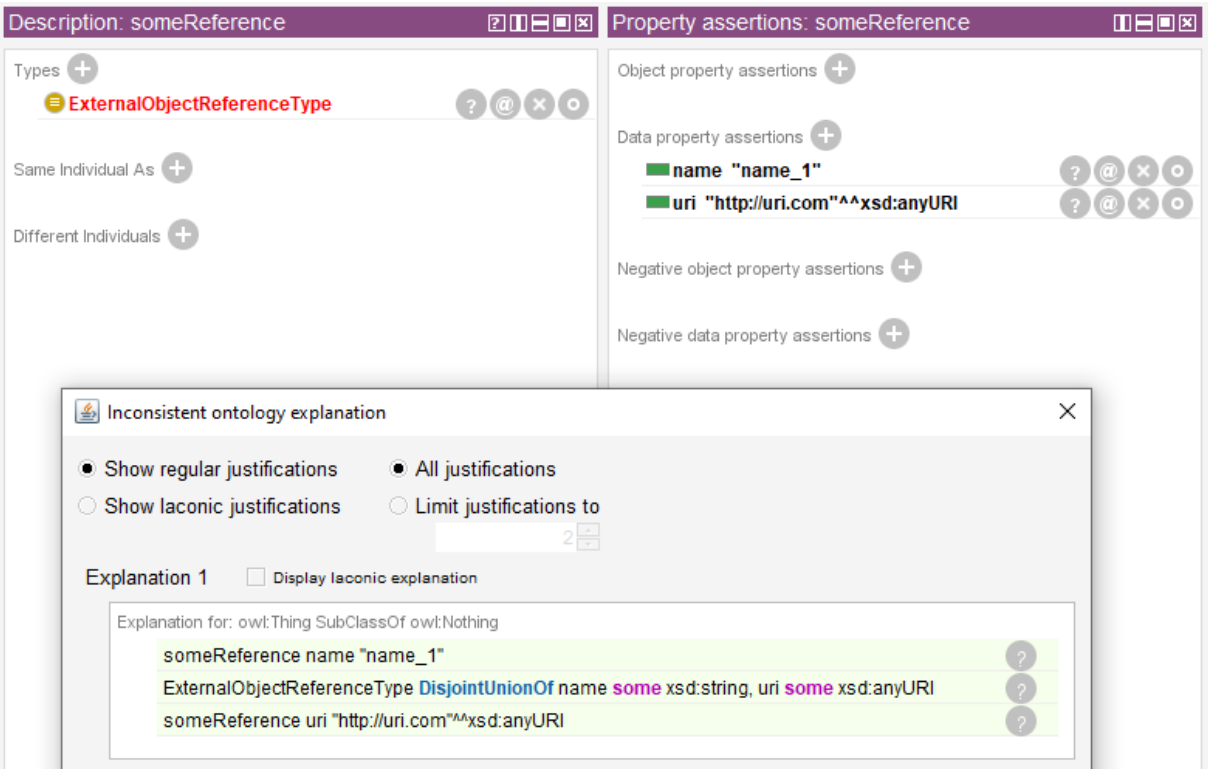

Figure 15. Reasoner explanation of inconsistent owl:DisjointUnionOf

In addition, there a native XML Schema element that is problematic when creating OWL properties, *xs:anyType*. In the CityGML application schema this type is used to allow extensible CityGML types and functionality, such as the *bldg:_GenericApplicationPropertyOfAbstractBuilding* abstract element of type *xs:anyType*. Consequently, to allow these elements to exist, a *xs:anyType* class is declared and any

elements of this type are declared as a subclass of it. If an extension to CityGML schema was created, their classes could be mapped as the *owl:equivalentClasses* of these proposed classes in order to integrate into the ontology. The usage of these classes is of course optional, as denoted in the application schema through the *minOccurs="0"* attribute. When mapping axiom restrictions, all mappings use *owl:someValuesFrom* which implies that the instance of the class contains at least one property of this type. These are paired with the mappings in [9], [14], to generate *owl:minCardinality* and *owl:maxCardinality* whenever the *minOccurs* or *maxOccurs* attribute is used to define a child element, allowing these instances to have an *owl:minCardinality* of 0 and thus keeping the class description conformed to the application schema.

One final consideration of the transformation process is the naming conventions of classes and properties. The transformation itself generates an intermediate identifier of the name of the entity appended to its namespace prefix. For example, the schema element in Fig. 13 would result with the attribute *rdf:about="gml:ExternalObjectReferenceType"*. Like in the XML to RDF pipeline a script is run after the transformation to fully qualify names into complete URIs. Similar to the naming conventions proposed in [11], the URIs used to name these entities are created from a the following pattern:

[domain]/[namespace prefix]#[object identifier] (3)

The names of transformed elements are appended to a predetermined domain, followed by the filename of the schema they were declared in – including each possible CityGML module – and then the name of the object itself. In the case of *rdf:about="gml:ExternalObjectReferenceType"*, the final output would be:

rdf:about=http://domain.uri/gml#ExternalObjectReferenceType (4)

An exception to this pattern occurs when the descendant *xs:elements* of *xs:complexTypes* are transformed into *owl:ObjectProperties* but reference an existing element instead of naming a new one. In order to differentiate the property from the referenced class, whenever an object property is created from the schema *'ref'* attribute, 'has' is added to the referenced name. In the case of the references to *gml:name* and *gml:description* made above, in Fig. 14, the final class identifiers would be:

rdf:about=http://domain.uri/gml#hasname (5)

rdf:about=http://domain.uri/gml#hasdescription (6)

These proposed strategies for XML schema to OWL transformation are an amalgam of previous transformation approaches with several specializations for the CityGML application schema. However, they are largely generalized and can be applied to schema outside of the main GML and CityGML schema, such as the external xAL addressing schema occasionally used by CityGML. The final step in converting CityGML into linked urban data, is the addition of the GeoSPARQL vocabulary and functionality into the ontology and converted instances.

2.3. Integrating GeoSPARQL

The addition of the official GeoSPARQL's vocabulary and ontology into these transformations is critical for providing interoperability as linked data. Since this approach features two distinct transformations for CityGML, the integration takes place in two parts of the pipeline: inside the XML instance to RDF

transformation based on the implementations of [11], [12] and in the postprocessing script after the XML schema to OWL transformation based on the suggestions in [4]. In these transformations several things must be taken into consideration such as the differences between the GML 3.2 ontology and the GML 3.1 application schema the transformations are based off and creating RDF instances that can be queried with GeoSPARQL functions.

Linking the generated CityGML ontology with GeoSPARQL is as straightforward as declaring two axioms. The class generated by *gml:_Geometry* is declared a subclass of *geo:Geometry* and the class generated by *gml:_Feature* is declared a subclass of *geo:Feature*. This allows all subclasses of *gml:_Feature* and *gml:_Geometry* to use GeoSPARQL properties. Features such as *blgd:Building* may implement the object property *geo:hasGeometry* to link to their respective geometries and geometries like *gml:Solid* can link to their GML representations with the datatype properties *geo:asGML*. Since GeoSPARQL endpoints can parse *geo:gmlLiterals* to perform spatial queries, the text stored in the original instance document can be reused to retain their geospatial information in OWL.
The official GeoSPARQL documentation states that:

> *Valid geo:gmlLiterals are formed by encoding geometry information as a valid element from the GML schema that implements a subtype of GM_Object … In GML 3.1.1 and GML 2.1.2 this is every element directly or indirectly in the substitution group of the element {http://www.opengis.net/ont/gml}_Geometry. [17]*

According to the resulting CityGML ontology this implies the following classes are either “directly or indirectly” in the substitution group of *gml:_Geometry* as shown in Fig. 16. During CityGML instance to RDF transformation, any element that is one of these classes and contains only ancestors of these classes, will retain a copy of their GML instance as a *geo:gmlLiterals* after transformation (fig. 17, 18).

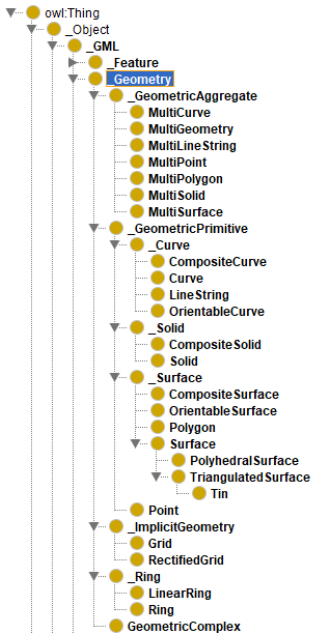

Figure 16. Generated CityGML ontology _Geometry classes

Additionally, if a particular coordinate system is used, it must be declared in the literal value as a *gml:srsName* attribute in order for GeoSPARQL endpoints to accurately parse the data. The default

reference system is *<http://www.opengis.net/def/crs/OGC/1.3/CRS84>*, thus any GML data that uses a different reference system must include it in the GML instances themselves or manually add it to the output *gml:gmlLiterals* after transformation. Through these processes a consistent strategy for generating geospatial linked data from CityGML instances can be implemented.

```
<bldg:BuildingPart gml:id="BU_69381AB243_1">
  <bldg:boundedBy>
    <bldg:RoofSurface gml:id="UUID_0ca316be-38cb-4c7f-8db7-723e08565df4">
      <bldg:lod2MultiSurface>
        <gml:MultiSurface gml:id="UUID_d4dad8ec-16ab-486c-ac31-acea2a7db390" srsDimension="3">
          <gml:surfaceMember>
            <gml:Polygon gml:id="UUID_f4ebea88-03cb-4bc7-85f3-645670657403">
              <gml:exterior>
                <gml:LinearRing gml:id="UUID_d5ce4476-b89d-424a-9e70-34ccf6ddc843">
                  <gml:posList>1841500.54989500 5175869.09632300 186.75556300 1841497.72348480
5175869.12983001 186.75556300 1841497.67131100 5175867.80319300 185.87431300 1841500.53329100
5175867.75429000 185.87431300 1841500.54989500 5175869.09632300 186.75556300 </gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </gml:surfaceMember>
        </gml:MultiSurface>
      </bldg:lod2MultiSurface>
    </bldg:RoofSurface>
  </bldg:boundedBy>
<bldg:BuildingPart gml:id="BU_69381AB243_1">
```

Figure 17. Sample CityGML bldg:Building instance

```
<owl:NamedIndividual rdf:about="http://domain.uri/LYON_1ER_BATI_2015#UUID_0ca316be-38cb-4c7f-
8db7-723e08565df4">
  <rdf:type rdf:resource="http://domain.uri/bldg#RoofSurface"/>
  <gml:id>UUID_0ca316be-38cb-4c7f-8db7-723e08565df4</gml:id>
  <bldg:lod2MultiSurface rdf:resource="http://domain.uri/LYON_1ER_BATI_2015#UUID_d4dad8ec-16ab-
486c-ac31-acea2a7db390"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="http://domain.uri/LYON_1ER_BATI_2015#UUID_d4dad8ec-16ab-486c-
ac31-acea2a7db390">
  <rdf:type rdf:resource="http://domain.uri/gml#MultiSurface"/>
  <gml:id>UUID_d4dad8ec-16ab-486c-ac31-acea2a7db390</gml:id>
  <gml:srsDimension>3</gml:srsDimension>
  <geo:asGML rdf:datatype="http://www.opengis.net/ont/geosparql#gmlLiteral">
      &lt;gml:MultiSurface xmlns:gml=http://www.opengis.net/gml
      xmlns="http://www.opengis.net/citygml/2.0"
      xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
      xmlns:core="http://www.opengis.net/citygml/2.0"
      xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance" gml:id="UUID_d4dad8ec-16ab-486c-ac31-acea2a7db390"
      srsDimension="3"&gt;&lt;gml:surfaceMember&gt;&lt;gml:Polygon gml:id="UUID_f4ebea88-03cb-
      4bc7-85f3-645670657403"&gt;&lt;gml:exterior&gt;&lt;gml:LinearRing gml:id="UUID_d5ce4476-
      b89d-424a-9e70-34ccf6ddc843"&gt;&lt;gml:posList&gt;1841500.54989500 5175869.09632300
      186.75556300 1841497.72348480 5175869.12983001 186.75556300 1841497.67131100
      5175867.80319300 185.87431300 1841500.53329100 5175867.75429000 185.87431300
      1841500.54989500 5175869.09632300 186.75556300
      &lt;/gml:posList&gt;&lt;/gml:LinearRing&gt;&lt;/gml:exterior&gt;&lt;/gml:Polygon&gt;&lt;/
      gml:surfaceMember&gt;&lt;/gml:MultiSurface&gt;
  </geo:asGML>
  <gml:hassurfaceMember rdf:resource="http://domain.uri/LYON_1ER_BATI_2015#surfaceMember_14"/>
</owl:NamedIndividual>
```

Figure 18. CityGML to RDF output of example bldg:RoofSurface

3. CONCLUSION

N-D geospatial data comes in many forms and can be difficult to analyze alongside other forms of urban data. There is a need for researchers and city planners to be able to study different forms of this data to understand the inherent relationships between them and how they evolve over time. The semantic web

and linked data directly respond to this problem by making data inherently interoperable through a single standard form and using the relationships between points of data as a core feature of the representation. This report highlights the major contributions of this effort: firstly, a study, analysis, and comparison of several tools and approaches available for the conversion of urban data into linked data formats; secondly, the implementation of a proof of concept tool to read and convert CityGML instances into multidimensional geospatial RDF data; thirdly, the implementation of a proof of concept tool (https://github.com/VCityTeam/UD-Graph) to generate an ontology constrained through cardinality, domain, range, and strongly typed class axioms as a semantic model of CityGML; finally, the implementation of initial geospatial queries to analyze the generated linked data that consider the structure of CityGML features. In addition, these contributions demonstrate the feasibility of integrating existing linked data standards such as GeoSPARQL.

However, there are several limitations of this proof of concept approach ranging from concepts not employed to areas that are lacking in performance. The first of which are the elements of XML schema not recognized by the transformation mapping patterns. Certain datatype elements such as *xs:list* and *xs:element* attributes such as *optional* and *abstract*, among others, are not taken into consideration during these transformations. Developing consistent transformation mappings for these could provide more accurate and rich geospatial linked data models and transformations. The model transformations also occasionally produce conflicting axioms depending on duplicate names are used to define properties, which occasionally occurs in the GML application schema. For instance, the *owl:DatatypeProperty gml:factor* is declared once as having an *rdfs:range* of *xs:integer* and again with a range of *xs:double*. When a reasoner is run, any classes that use this property in an *owl:someValuesFrom* axiom are inferred to be equivalent to *owl:Nothing*, as these ranges conflict. A modification could be made to the model post-processing script that consolidates these axioms with an *owl:UnionOf* statement.

Additionally, the instance post processing activity itself is time-consuming especially when converting a breadth heavy XML tree. This means performing “live” data conversion is not feasible and would require further optimizations to either the transformations or the scripts themselves. Furthermore, the representation in RDF takes up a considerable amount of storage space compared to the original GML format. It may be possible to reduce the size of the generated instances and ontology by utilizing a different RDF friendly format such as Turtle. Although these issues do not prevent the proof of concept results from acting as a valid approach to transforming CityGML into linked data, they should be considered in any future work based on this proposed approach.

ACKNOWLEDGEMENTS

We would like to thank LIRIS UMR 5205 for funding through the action transversale (LIRIS AT 2019-2021). We would also like to thank Hamida Seba of the GOAL team for this fruitful collaboration among the BD, GOAL, and Origami teams within LIRIS. In addition, we thank the members of the Virtual City Project for their valued advice and support.

REFERENCES

[1] L. Brink, “Geospatial Data on the Web,” Oct. 2018. https://www.ncgeo.nl/index.php/en/publicatiesgb/publications-on-geodesy/item/2789-geospatial-data-on-the-web (accessed Jul. 23, 2020).

[2] A.-H. Hor, M. Jadidi, and G. Sohn, “BIM-GIS INTEGRATED GEOSPATIAL INFORMATION MODEL USING SEMANTIC WEB AND RDF GRAPHS,” Jul. 2016, vol. III–4, pp. 73–79, doi: 10.5194/isprs-annals-III-4-73-2016.

[3] “About OGC | OGC.” https://www.ogc.org/about (accessed Jul. 23, 2020).

[4] R. Battle and D. Kolas, “Enabling the geospatial Semantic Web with Parliament and GeoSPARQL,” *Semantic Web*, vol. 3, no. 4, pp. 355–370, Jan. 2012, doi: 10.3233/SW-2012-0065.

[5] E. Hietanen, L. Lehto, and P. Latvala, “PROVIDING GEOGRAPHIC DATASETS AS LINKED DATA IN SDI,” *ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XLI-B2, pp. 583–586, Jun. 2016, doi: 10.5194/isprs-archives-XLI-B2-583-2016.

[6] R. L. G. Lemmens, G. Falquet, and C. Métral, “Towards Linked Data and ontology development for the semantic enrichment of volunteered geo-information.,” *Proc. Link-VGI Link. Anal. Volunt. Geogr. Inf. VGI Differ. Platf. Workshop AGILE 2016 Conf.*, 2016, Accessed: Jul. 23, 2020. [Online]. Available: https://research.utwente.nl/en/publications/towards-linked-data-and-ontology-development-for-the-semantic-enr.

[7] O. Zalamea, J. Orshoven, and S. Thérèse, “From a CityGML to an ontology-based approach to support preventive conservation of built cultural heritage.,” Jun. 2016.

[8] T. R. Kramer, B. H. Marks, C. I. Schlenoff, S. B. Balakirsky, Z. Kootbally, and A. Pietromartire, “Software Tools for XML to OWL Translation,” Jul. 2015, Accessed: Jul. 23, 2020. [Online]. Available: https://www.nist.gov/publications/software-tools-xml-owl-translation.

[9] H. Bohring and S. Auer, “Mapping XML to OWL ontologies,” Jan. 2005, pp. 147–156.

[10] C. Métral and G. Falquet, “EXTENSION AND CONTEXTUALISATION FOR LINKED SEMANTIC 3D GEODATA,” in *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Sep. 2018, vol. XLII-4-W10, pp. 113–118, doi: https://doi.org/10.5194/isprs-archives-XLII-4-W10-113-2018.

[11] L. Brink, P. Janssen, W. Quak, and J. Stoter, “Linking spatial data: automated conversion of geo-information models and GML data to RDF,” *Int. J. Spat. Data Infrastruct. Res.*, vol. 9, pp. 59–85, Oct. 2014, doi: 10.2902/1725-0463.2014.09.art3.

[12] M. S. Bekatoros and M. Koubarakis, “A web-based GML to stRDF / GeoSPARQL conversion tool,” p. 99, Feb. 2015.

[13] M. Ferdinand, C. Zirpins, and D. Trastour, “Lifting XML schema to OWL,” Jul. 2004, vol. 3140, pp. 354–358, doi: 10.1007/978-3-540-27834-4_44.

[14] I. Bedini, C. Matheus, P. F. Patel-Schneider, A. Boran, and B. Nguyen, “Transforming XML Schema to OWL Using Patterns,” in *2011 IEEE Fifth International Conference on Semantic Computing*, Sep. 2011, pp. 102–109, doi: 10.1109/ICSC.2011.77.

[15] L. Brink, P. Janssen, and W. Quak, “From Geo-data to Linked Data: Automated Transformation from GML to RDF,” *Linked Open Data - Pilot Linked Open Data Ned. Deel 2 - Verdieping Geonovum 2013 Pp 249-261*, 2013, Accessed: Jul. 23, 2020. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A8ec77e83-8406-47d3-8705-32633619ba1f.

[16] C. Tsinaraki and S. Christodoulakis, “XS2OWL: A Formal Model and a System for Enabling XML Schema Applications to Interoperate with OWL-DL Domain Knowledge and Semantic Web Tools,” Jan. 2007, vol. 4877, pp. 124–136, doi: 10.1007/978-3-540-77088-6_12.

[17] M. Perry and J. Herring, “OGC GeoSPARQL - A Geographic Query Language for RDF Data,” p. 75, 2012.