FROM GEO-DATA TO LINKED DATA: AUTOMATED TRANSFORMATION FROM GML TO RDF

Linda van den Brink, Paul Janssen
Geonovum

Wilko Quak
TU-DELFT

Linked data provide an alternative route for dissemination of spatial information as compared to the traditional SOA-based SDI approach. Where the latter is built on predefined structuring of semantics within domains, linked data is open to linking information to any data over the Web. In this respect both are complementary. The traditional approach providing a mechanism for a basis of standardized and structured data within domains, and linked data providing an open mechanism for sharing and combining. GML as the ISO standard for exchange of service based spatial data and RDF as the linked data format are therefore related. GML provides the format in which many spatial datasets are available and exchanged. This standardization process and effort has been realized on a large scale. Why not let the web of linked data take advantage of this effort? This article will focus on the use of GML structured data as a source for deriving RDF structured data.

The first part of the paper focusses on deriving linked data from GML data. The first version of GML, v1.0, was based on RDF. From version 2.0 onwards GML was based on XML and XML Schema, but the object-property structure was retained. We describe a transformation for translating any correctly structured GML to RDFS/OWL automatically, using XSLT. Because GML's object-property structure translates very well to triples, the transformation is straightforward. Well-known GML content elements such as names and descriptions are mapped to their RDF equivalent. However, any semantics specific to the input GML data (a.k.a. the application schema) are ignored in this translation.

In the second part, we study how more meaningful RDF can be created from GML, given the underlying information model, by transforming it from UML to RDFS/OWL. There exists a straightforward mapping to convert a UML model into a RDFS/OWL vocabulary. However, the re-use of existing concepts in vocabularies takes a central role in RDFS/OWL while in UML the use of vocabularies is not supported. We describe how annotating the UML model could improve this translation.

Background

Linked data is the new kid on the block in the set of standards relevant for geographic information. It provides an alternative route for dissemination of spatial information as compared to the already considered traditional SOA-based SDI approach. Basically the difference is about flexibility and openness. Where the latter is built on predefined structuring of semantics within domains, linked data is open to linking information to any data over the Web. In this respect it much more appeals to the web 3.0 philosophy: unique information features that are there, floating around, and can be accessed and or extended any time by anyone for any purpose. But what does that mean to what has been done and realized in the spatial information infrastructure until now? We state that it is complementary. The traditional approach providing a mechanism for a basis of standardized and structured data within domains, and linked data providing an open mechanism for sharing and combining. The traditional approach is characterized by a service based dissemination of GML

structured data. In that approach data specifications provide clear definitions of semantics in predefined domains and use cases. These are implemented in XML schema, providing a well defined and verifiable means of information exchange. The strong point of it is that the proper purpose of standardization and harmonisation, being interoperability can be addressed through agreement and sharing of vocabulary. Once agreed the requirements and rules for communication are set and can be implemented in a verifiable way. The quality of implementation can be measured and therefore managed. But there is a downside: the vocabulary, semantics are defined within information domains. Resulting in predefined information silos each related to different information domains. Within the silo interoperability is assured by shared and foreseen concepts, but between silos little harmonization takes place, and for not yet foreseen concepts and relations the structure is too rigid. This is exactly the weak spot where linked data can be of help.

In a spatial data infrastructure GML data are generated and served from different feature based sources. Generally transformation services will do the transformation from these local sources to the GML structured data. In many SDI projects and programmes a lot of effort has been put into that activity. For linked data, including the related RDF format and GeoSPARQL, a similar way can be followed. Transformation services acting on local sources and generating GeoSPARQL endpoints. But why not reuse the already existing GML sources? Since these are already structured in a standardized and defined way, RDF transformation can be standardized as well. Linked date can therefore build on the structure already provided. One would expect a simple rule of benefit: profiting twice by reusing work that has been done once. The challenge therefore is to investigate this way of generating linked data out of GML. In the process we will come to understand more about principal differences between linked data and GML and their complementary roles.

The following diagram depicts several ways of building linked data on top of an SDI.

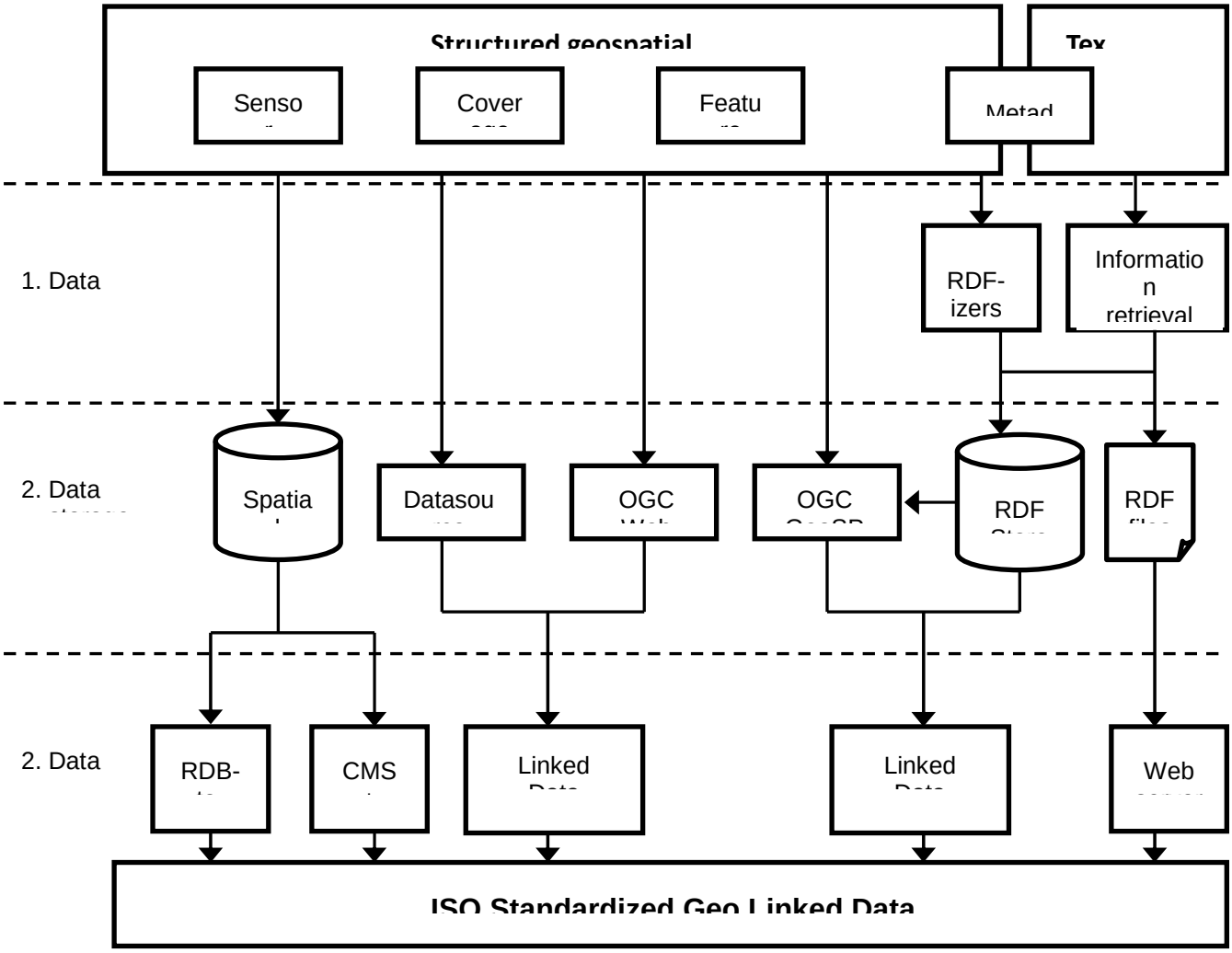

Figure 1: RDF as part of SDI

Principal ways of integrating linked data in a spatial data infrastructure [1].

Related to the diagram GML to RDF transformation can be considered in the RDF-izer part. The Linked Data Wrapper is also interesting but for this article out of scope.

Deriving linked data from GML

Geography Markup Language is a standard for the storage and transport of geographic information. The first version of GML, v1.0 [2], was published in May of the year 2000. Key concepts in the GML model of the world are the *feature*: an abstraction of a real world phenomenon, the *geographic feature*: a feature which is associated with a location relative to the Earth, and the *feature collection*, a collection of features that can itself be regarded as a feature and that gives a digital representation of the real world. Features have properties, geographic features have properties whose value may be a geometry.

GML 1.0 used a geometry model called “Simple Features”, with definitions for point, line string, polygon, and some other basic geometric shapes. In addition it provided a coordinates element for encoding coordinates, and a Box element for defining extents. In its simplest form, GML contains no more semantics than this: geographic features with associated geometric shapes. The standard, however, includes an extension mechanism which makes it possible to define application-specific extensions with added semantics, for example distinct object classes for River and Road, each with their specific properties.

GML 1.0 described three encoding profiles for geographic features, two of which were based on XML (Extensible Markup Language) and DTD (Document Type Definition: similar to XML Schema which was not used because it was not yet standardized at that time), while the third was based on RDF and RDF Schema. This means that it was possible to write GML 1.0 as an RDF file! OGC intentionally created the model of GML as consistent with RDF. Like RDF, the model of GML was basically a set of triples. GML features have properties; each property is a {name, type, value} triple. Properties can have either simple values or have a geometry object as value ("geometric value"). According to the naming conventions of GML, object properties always had names starting with a lowercase letter, while GML object classes had names starting with an uppercase letter. All objects had an optional ID which could be used together with the GML document URI as a fragment identifier. In this manner, GML objects could be referenced as resource – very much the linked data way of working!

GML 1.0 example, in which "yourhouse" and "myhouse" have the same location:

```
<Building ID = "yourhouse" .. >
      <location>
            <Point ID = "134">
                  <coordinates>
                        2455.12, 3443.78
                  </coordinates>
            </Point>
      </location>
</Building>

<Building ID = "myhouse" .. >
      <location>
            <Point resource = "#134" />
      </location>
</Building>
```

From version 2.0 onwards GML was based on XML and XML Schema, and the RDF profile was no longer used. But an interesting fact is that the object-property structure, in which objects have properties and properties have either simple values or objects as values - basically a triple structure – always stayed, up until the latest version, 3.3 [3]. And because GML and RDF both have a triple structure, it is easy to define a transformation for translating any correctly structured (that is, conformant to the object-property /triple structure) GML data to RDFS/OWL automatically. As an experiment, we implemented such a transformation using XSLT 2.0 [4]. Well-known GML content elements such as names and descriptions are mapped to their RDF equivalent. Objects and properties are recognized based on their place in the triple structure and are transformed accordingly.

The experimental implementation has 8 templates; counting whitespace and comments it has 88 lines. This shows the simplicity of the transformation. The full XSLT stylesheet is included as an appendix. The stylesheet was tested on a sample GML file containing land use plans, conforming to the Dutch standard IMRO[1].

Workings of the transformation

```
<xsl:template match="/">
    <rdf:RDF>
      <xsl:apply-templates select="//*[count(ancestor::*) mod 2 = 0]"/>
    </rdf:RDF>
</xsl:template>
```

Figure 2: Sample XSLT fragment

The transformation starts at the top of the GML file and selects all features, even the ones that are nested as property value of another feature. The features can be recognized because they always have an even number of ancestors (levels in the XML hierarchy). The GML file starts with a feature (usually a feature collection), which

[1] (Information Model Ruimtelijke Ordening - spatial planning)

has properties, which in turn have features as values. A simple filter can take advantage of this fact. Those elements that have an even number of ancestors (levels in the XML hierarchy) are transformed to rdf:Description elements. The `rdf:about` attribute is filled with `gml:id` if it's present; if not, an id is generated.

Well-known, standardized GML properties are transformed to an appropriate property. When possible a standard property from RDF or RDFS is used. For example, gml:description is transformed to rdfs:comment, gml:name to rdfs:label. Properties that are not known (i.e. not standard GML, but from some domain-specific extension) are not changed, but receive the same name in the RDF output.

Properties with nested content (they have a feature as value, which is not referenced but included directly) receive special treatment. The nested feature is already recognized, and transformed to an rdf:Description, by the first templates. The property with nested content is transformed to a property that *references* the feature that was nested, using an rdf:resource attribute containing the id of the feature prefixed with a hash '#'. Usually an id is not present in these cases, and one is generated automatically.

Properties that link to a feature in the GML are transformed to their RDF equivalent, an RDF property with an `rdf:resource` attribute containing the id of the referenced feature.

Geometry

Geometry is encoded in GML as objects with properties, so it has the same triple structure. This structure is translated directly to RDF, in the same way as described above for the other features. But this is not the most useful way to represent geometry in RDF. Because geometries like curves and polygons are heavily nested structures in GML, it takes a lot of resources to represent them in RDF. A sample line geometry:

```
<gml:Curve>
    <gml:segments>
        <gml:LineStringSegment>
            <gml:posList>187453.376 429000.792 187444.313 429004.534</gml:posList>
        </gml:LineStringSegment>
        <gml:LineStringSegment>
            <gml:posList>187444.313 429004.534 187442.564 428999.999 187441.651 428997.63</gml:posList>
        </gml:LineStringSegment>
        <gml:LineStringSegment>
            <gml:posList>187441.651 428997.63 187453.094 428994.092</gml:posList>
        </gml:LineStringSegment>
        <gml:LineStringSegment>
            <gml:posList>187453.094 428994.092 187456.661 428992.989 187458.431 428992.407</gml:posList>
        </gml:LineStringSegment>
        <gml:LineStringSegment>
            <gml:posList>187458.431 428992.407 187459.987 428998.063</gml:posList>
        </gml:LineStringSegment>
        <gml:LineStringSegment>
            <gml:posList>187459.987 428998.063 187453.376 429000.792</gml:posList>
        </gml:LineStringSegment>
    </gml:segments>
</gml:Curve>
```

Figure 3: Line geometry in GML

```
<rdf:Description rdf:about="d1e67458" rdf-type="http://someuri#Curve">
   <gml:segments rdf:resource="#d1e67460"/>
   <gml:segments rdf:resource="#d1e67463"/>
   <gml:segments rdf:resource="#d1e67466"/>
   <gml:segments rdf:resource="#d1e67469"/>
   <gml:segments rdf:resource="#d1e67472"/>
   <gml:segments rdf:resource="#d1e67475"/>
</rdf:Description>
<rdf:Description rdf:about="d1e67460" rdf-type="http://someuri#LineStringSegment">
   <gml:posList>187453.376 429000.792 187444.313 429004.534</gml:posList>
</rdf:Description>
```

Figure 4:The same Line geometry (fragment) in RDF

The example shows a curve with several nested segments in GML. In RDF these nested segments become links to these segments as separate Description resources (only the first segment, with id 'd1e67460' is shown). This becomes even more complex with polygons that have patches, interior and exterior rings, which are built up from curves, etc. This way of encoding geometry makes location-based querying the RDF very hard. In this short experiment there was no time to look at different, easier ways of encoding the geometry in RDF, but several possibilities exist. These alternatives range from very simple solutions, such as Basic Geo [5], usable for representing latitude and longitude using WGS84 as reference datum; to more full-fledged solutions like GeoSPARQL [6], which allows a WKT (Well Known Text) serialization and a GML serialization. Which of these to use is a very relevant question we must answer before starting to create geo-linked data on a larger scale.

More semantics

The XSLT stylesheet described above transforms GML data to RDF in a generic way, based on GML's object-property structure. But it ignores any domain-specific semantics the GML may have. The IMRO sample file has a lot of domain-specific semantics, defined in the IMRO GML application schema:

```
<imro:featureMember>
    <imro:Bouwaanduiding gml:id="NL.IMRO.0268.ID101733-00">
        <imro:identificatie>NL.IMRO.0268.ID101733-00</imro:identificatie>
        <imro:typePlanobject>bouwaanduiding</imro:typePlanobject>
        <imro:plangebied xlink:href="#NL.IMRO.0268.BP5000-VG01"/>
        <imro:naam>onderdoorgang</imro:naam>
        ...
```

Figure 5: Fragment IMRO GML

imro:Bouwaanduiding (building indication) is translated to an rdf:Description of rdf:type http://someuri#Bouwaanduiding. Instead of 'someuri' this should refer to some location where the IMRO ontology is published. All properties of imro:Bouwaanduiding are transformed to RDF properties of the same name (see Figure 6). These should all be defined in the IMRO ontology (which we do not have, at least not in RDF/OWL at this stage). Some of the properties could be mapped to Linked Data vocabularies. For example, it would be appropriate to translate imro:naam to rdfs:label, but this is not known to the transformation, as it is a generic tool and is not aware of the meaning of the IMRO vocabulary.

```
<rdf:Description rdf:about="NL.IMRO.0268.ID101733-00"
                 rdf-type="http://someuri#Bouwaanduiding">
   <imro:identificatie>NL.IMRO.0268.ID101733-00</imro:identificatie>
   <imro:typePlanobject>bouwaanduiding</imro:typePlanobject>
   <imro:plangebied rdf:resource="#NL.IMRO.0268.BP5000-VG01"/>
   <imro:naam>onderdoorgang</imro:naam>
   <imro:labelInfo rdf:resource="#d1e17"/>
   <imro:geometrie rdf:resource="#d1e33"/>
   <imro:aanduiding rdf:resource="#NL.IMRO.0268.ID44435-00"/>
</rdf:Description>
```

Figure 6: Same fragment; IMRO RDF

This aspect must be addressed because usually the GML is extended for a certain domain. It contains rich semantics, which would be lost in the translation to RDF: in the context of the semantic web these semantics are of course crucial! These semantic extensions are described in a standardized way in a so-called GML application schema. For the Dutch IMRO standard such an application schema is available. Therefore not only the GML, but also the GML application schema should be translated to Linked Data standards. Also, domain-specific knowledge about the application schema could improve the mapping, taking into account established Linked Data languages and vocabularies like RDF and RDFS, but also for example Dublin Core or SKOS. In our experiments we looked at this, and the next section describes some interesting aspects on the translation to RDF of specific semantics from an application-specific GML structure like IMRO.

Creating meaningful RDF from Geo-information models

Meaning in the semantic web comes from vocabularies and the method in the previous section does not provide or use a vocabulary. By using (or creating) a vocabulary for IMRO the mapping from GML to RDF would more useful. Such a vocabulary can be automatically derived from the IMRO information model. The IMRO information model is available as UML diagram which is then automatically converted into a GML application schema. Now there are two options to generate an OWL vocabulary. First to derive it from the UML model directly, second to derive it from the GML application schema. The first option has the advantage that UML is more mainstream IT than GML application schema and that a well-defined mapping from UML to OWL is defined by the OMG [7]. The second option (mapping from the GML application schema) has the advantage that it is spatially aware (since a GML application schema has well defined spatial semantics) which would result in a better mapping for spatial objects. A combination of both would be best and this can be achieved by defining specific mappings from UML for spatial modelling constructs. Currently these mappings are partially stable: for spatial datatypes (OGC simple Features) a mapping is described in [6]. How to map UML stereotypes used in spatial models (such as <<FeatureType>>) is still under development. Shapechange [8] implements an experimental version of these mapping rules and it has been successfully applied to the IMRO model resulting in a IMRO vocabulary (Figure 7). By slightly adapting the GML2RDF script it is possible to generate IMRO RDF that refers to the IMRO vocabulary.

```
<Class xmlns="http://www.w3.org/2002/07/owl#" rdf:about="http://www.geonovum.nl/imro2008#Bouwaanduiding">
  <subClassOf xmlns="http://www.w3.org/2000/01/rdf-schema#"
    rdf:resource="http://www.geonovum.nl/imro2008#Aanduiding"/>
  <subClassOf xmlns="http://www.w3.org/2000/01/rdf-schema#">
    <Restriction xmlns="http://www.w3.org/2002/07/owl#">
      <onProperty rdf:resource="http://www.opengis.net/geosparql#defaultGeometry"/>
      <allValuesFrom rdf:resource="http://www.opengis.net/geosparql#Surface"/>
    </Restriction>
  </subClassOf>
</Class>
```

Figure 7: IMRO Bouwaanduiding vocabulary entry as generated by ShapeChange

By using the automatically generated IMRO vocabulary one harmonizing aspect of RDF is not used: no existing RDF vocabularies are used. So imro:naam would get an entry in the IMRO vocabulary where it would be more meaningful to map it to rdfs:label. However the knowledge that imro:naam is in fact an rdfs:label is not available in the UML model and cannot be automatically mapped. In order to improve the UML model for better mapping to RDF one could extend the UML model by annotating the UML attributes that have a special meaning in RDF with a link to their RDF counterpart. If, for example, the imro:naam attribute in the UML model would be get the following annotation (via a tagged value): ‘rdfVocabulary=rdfs.label’ it would be possible to make an optimal link between UML and RDF. We plan to make a proposal for the extension of UML modeling.

References:

[1] Francisco .J. Lopez-Pellicer, Luis M. Vilches-Blazquez, F.Javier Zarazaga-Soria, Pedro R. Muro-Medrano, O. Corcho, THE DELFT REPORT: LINKED DATA AND THE CHALLENGES FOR GEOGRAPHIC INFORMATION STANDARDIZATION – and also An RM-ODP Enterprise View for Spatial Data Infrastructures, Revista Catalana de Geografía. 2012, vol. XVII, nº 44. ISSN 1988-2459.

[2] Lake, R. and Cuthbert, A. Geography Markup Language (GML) v1.0. OpenGIS® Consortium (OCG), 12 May 2000. http://portal.opengeospatial.org/files/?artifact_id=7197

[3] Portele, C. OGC® Geography Markup Language (GML) — Extended schemas and encoding rules. Open Geospatial Consortium, 7 February 2012. https://portal.opengeospatial.org/files/?artifact_id=46568

[4] Kay, M. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium (W3C), 23 January 2007. http://www.w3.org/TR/xslt20/

[5] Brickley, D. Basic Geo (WGS84 lat/long) Vocabulary. W3C Semantic Web Interest Group. http://www.w3.org/2003/01/geo/

[6] Perry, M. and Herring, J. OGC GeoSPARQL - A Geographic Query Language for RDF Data. Open Geospatial Consortium, 10 September 2010. http://www.opengeospatial.org/standards/geosparql

[7] Object Management Group, Ontology Definition Metamodel (version 1.0) http://www.omg.org/spec/ODM/1.0/

[8] ShapeChange, Processing application schemas for geographic information. http://www.shapechange.net/

Appendix- XSLT stylesheet GML > RDF

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:imro="http://www.geonovum.nl/imro/2008/1"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:math="http://www.w3.org/2005/xpath-functions/math"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    exclude-result-prefixes="xs"
    version="2.0">

    <xsl:output indent="yes"/>
    <xsl:strip-space elements="*"/>
```

<!-- root template
This template matches the root (‘/’) of the GML file and includes an apply-templates instructions which causes all elements present in the GML file (‘//*’) that have an even number of ancestors (the filter count(ancestor::*) mod 2 = 0), to be processed. Those elements that have an even number of ancestors (levels in the XML hierarchy) are the Classes (this follows from the GML Object-property pattern). -->

```
    <xsl:template match="/">
        <rdf:RDF>
          <xsl:apply-templates select="//*[count(ancestor::*) mod 2 = 0]"/>
        </rdf:RDF>
    </xsl:template>
```

<!-- template for features / resources.
This template matches all elements that have an even number of ancestors: the Classes. These are transformed to rdf:Description. The rdf:about attribute is filled with gml:id if it’s present; if not, an id is generated. @srsName and the properties are then processed (apply-templates). -->

```
    <xsl:template match="*[count(ancestor::*) mod 2 = 0]">
        <rdf:Description rdf:about="{if (@gml:id) then @gml:id else generate-id(.)}" rdf-
type="http://someuri#{local-name()}">
            <xsl:apply-templates select="@srsName"/>
            <xsl:apply-templates/>
        </rdf:Description>
    </xsl:template>
```

<!-- template for srsName GML property
This template matches the srsName attribute and transforms this to an RDF property named gml:srsName. The URN that refers to the coordinate reference system is contained in rdf:resource. -->

```
    <xsl:template match="@srsName">
        <gml:srsName rdf:resource="{.}"/>
    </xsl:template>
```

<!-- template for description GML property
This template matches the gml:description element and transforms this to rdfs:comment. -->

```
    <xsl:template match="gml:description">
        <rdfs:comment><xsl:value-of select="text()"/></rdfs:comment>
    </xsl:template>
```

<!-- template for name GML property

This template matches the gml:name element and transforms this to rdfs:label. -->

```
    <xsl:template match="gml:name">
        <rdfs:label><xsl:value-of select="text()"/></rdfs:label>
    </xsl:template>
```

<!-- template for properties with simple values
This template matches all elements (*) that have an uneven number of hierarchy levels/ancestors (ancestor::*) mod 2 != 0) and no further hierarchy levels nested inside (not(child::*)). These are the simple properties. They are transformed to RDF properties. -->

```
    <xsl:template match="*[count(ancestor::*) mod 2 != 0 and not(child::*)]">
        <xsl:element name="{name()}">
            <xsl:value-of select="text()"/>
        </xsl:element>
    </xsl:template>
```

<!-- template for properties with nested object as content
This template matches all elements (*) that have an uneven number of hierarchy levels/ancestors (ancestor::*) mod 2 != 0) but DO have further hierarchy levels nested inside (child::*). These nested children are GML Objects (as follows from the Object-property pattern) and are therefore transformed to Classes as child of rdf:RDF by the Class Template.
This template creates a property for each of these nested Objects (xsl:element name="{parent::*/name()}") and an rdf:resource attribute (xsl:attribute name="rdf:resource") which points to the Class representing the Object either using its gml:id or a generated id (concat('#', if (@gml:id) then @gml:id else generate-id(.))). -->

```
    <xsl:template match="*[count(ancestor::*) mod 2 != 0 and child::*]">
        <xsl:for-each select="*">
            <xsl:element name="{parent::*/name()}">
               <xsl:attribute name="rdf:resource" select="concat('#', if (@gml:id) then
@gml:id else generate-id(.))"></xsl:attribute>
           </xsl:element>
        </xsl:for-each>
    </xsl:template>
```

<!-- template for properties with links to other objects
This template matches all elements (*) that have an uneven number of hierarchy levels/ancestors (ancestor::*) mod 2 != 0) and contain an xlink reference to another element in the GML file (normalize-space(@xlink:href)). These are transformed to an RDF property with an rdf:resource attribute containing the reference (xsl:attribute name="rdf:resource" select="@xlink:href"). -->

```
    <xsl:template match="*[count(ancestor::*) mod 2 != 0 and normalize-
space(@xlink:href)]">
        <xsl:element name="{name()}">
            <xsl:attribute name="rdf:resource" select="@xlink:href"/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```