

Supporting GI standards with a model-driven architecture

Roy Grønmo

SINTEF Telecom and Informatics

Forskningsveien 1, Pb 124 Blindern

N-031 4 Oslo, Norway

T+47 22 06 74 32

roy.gronmo@informatics.sintef.no

ABSTRACT

This paper describes an approach to generate GI standards such as OpenGIS Geography Markup Language (GML) and ISO 19 118 XML Encoding from UML. GML application schemas in OpenGIS context are being developed by using XML Schema directly and without any relation to UML models. It is SINTEF's belief that a model-driven architecture using UML is more suitable for specifying GML application schemas. SINTEF has used a model-driven architecture for several years and has also been strongly involved in defining the ISO GI standards. The model-driven architecture is investigated further by SINTEF in the project GeNorway - Model-based infrastructure for living geospatial data in eNorway. GeNorway shall implement a Norwegian version of an OGC-conform Web Mapping test case built on top of the Norwegian Spatial Data Infrastructure (NGIS).

Categories and Subject Descriptors

H.1.0 [Models and Principles]: General - *model-driven architecture, code generation.*

General Terms

Design, Standardization.

Keywords

ISO/TC 211, OGC, GML, GIS, UML, XML.

1. INTRODUCTION

This paper describes a model-driven architecture (MDA) [13] to generate OpenGIS Geography Markup Language (GML) [10] from UML. ISO/TC 211 [7] recommends the MDA approach in general and specifically as a way of reaching XML specifications by the standards ISO 19103 Conceptual schema language [3], ISO 19109 Rules for application schema [4] and ISO 19118 XML Encoding [5].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GIS'01, November 9-10, 2001, Atlanta, Georgia, USA.

Copyright 2001 ACM 1-581 13-443-6/01/0011...\$5.00.

With the ISO approach these steps are needed to achieve an XML application schema:

- Define a conceptual model with UML.
- Define mapping rules from UML to XML Schema.
- Implement these mapping rules by code generation tools.

SINTEF has been involved in specifying the ISO standards as well as implementing MDA tools in GI projects such as DISGIS [1] and JNIP [2]. These efforts are continued within the project GeNorway - Model-based infrastructure for living geospatial data in eNorway. GeNorway is an ongoing two-year project partially funded by the Norwegian Research Council and consists of five members: Bærum Municipality, Quadri Components, Norkart, the Norwegian Mapping Authority and SINTEF. The project shall try out the ISO GI and OGC standards and implement an OGC conform Norwegian version of Web Mapping technology. In order to do this the Norwegian Mapping Authority is committed to redefine the proprietary Norwegian data catalogue format, SOSI, as UML models according to the ISO/TC 211 standards. This will result in several hundred UML models. Code generation tools will greatly reduce the costs needed to bring these UML models further to ISO and OGC conform specifications and implementations. None of these UML models are currently defined. Therefor GeNorway has started out to develop prototype tools for a chosen test case. This work is presented as proof-of-concept in this paper, and intends to be a good road map for the continuous NGIS and GeNorway work. The goal of the test case was to:

- determine if GML could be fully generated from UML;
- develop code generation tools that can be extended to fully generate GML;
- contribute to bridging the gap between the ISO GI and OGC standardisation.

This paper follows the same structure as a recommended model-driven approach. In Section 2 a conceptual UML model is defined. Section 3 defines the needed mapping rules from the conceptual model to GML. Section 4 identifies two alternative code generation architectures that could implement the mapping rules of section 3. The chosen alternative is shown in more detail in section 5. Section 6 covers an important building block of the chosen code generation approach and how it further improves two previous approaches. Section 7 finally shows that the UML model defined in section 2 may be mapped to both GML and ISO 19118 XML.

2. Conceptual UML model

It was decided to use the city example of GML2.0 [1] as the test case. This involves some of the typical model aspects, making it suitable for making a prototype. Another important advantage is that we can use the GML schema included with the example, as a correct answer to what the code generation tool should generate. This section discusses how to develop a conceptual UML model for the city example.

The first step of the MDA methodology is to specify a UML model. This model should ideally be a conceptual model that hides implementation details so that the model is easy to understand. The UML model shall be defined as a UML class diagram. There are at least three possible approaches to specifying a UML model for the city example:

1. *UML figure directly from the city example:* use the UML figure that comes along with the city example of the GML specification.
2. *Conceptual model with GML terms:* make good conceptual UML models with the terms that are defined in the feature and geometry specifications of GML.
3. *Conceptual model with ISO terms:* follow the ISO 19109 Rules for application schema and ISO 19103 Conceptual schema language and use terms from the other ISO/TC 211 standards-to-be whenever applicable. Note that good conceptual models are achieved by following these ISO specifications.

Approach 1 was rejected due to the condition of the UML figures in the GML2.0 specification. The figures are too tightly coupled with the actual encoding format to be good conceptual models, and the figures have some constructions that are rather confusing as well as being illegal UML constructions. Approach 2 may be a good idea, but is demanding, as there are no guidelines for how to do this. It did not fit within the available timeframe of this test case. Approach 3 gives good conceptual UML models.

The test case tries out approach 3. A major advantage of this choice is that the guidelines for defining such UML models are stated by the ISO/TC 211 standards. Aid in defining ISO conform UML models is supported by SINTEF developed scripts [1] [2]. Another advantage is that this approach is a bridge between ISO and OGC. A disadvantage is that the terminology at the UML model level does not correspond with the terminology at the XML implementation level. For instance GM-Curve is the correct term at UML model level, while LineStringPropertyType is the term at XML implementation level.

The model elements of the city example are taken directly from the GML specification, which again is a very simplified model of a real-life city model. A CityModel contains a set of cityMembers. The cityMembers may be of two different feature types: Road or River. Both Road and River have one-dimensional geometry represented by the ISO type GM-Curve. A Mountain feature type is included in the example to demonstrate a feature that cannot be a cityMember. The resulting ISO UML model of the GML city example is shown in Figure 1.

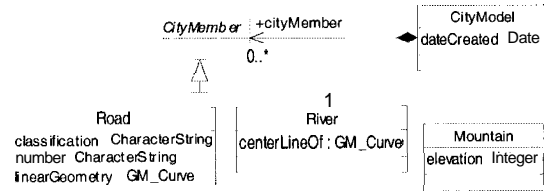


Figure 1 UML model following ISO rules for the city model

3. UML to GML Mapping Rules

This section describes the mapping rules from UML to GML2.0 that were needed for the city example. UML classes are mapped to three different GML types depending on the kind of relations the class is involved in:

- *GML type FeatureCollection.* UML classes that have at least one aggregation or composition association to another class. For the city example this rule applies to CityModel.
- *GML type FeatureMember.* UML classes that are aggregated or composed into an association by at least one class. For the city example this rule applies to CityMember.
- *GML type Feature.* All other classes that do not fall into the two categories above. For the city example this rule applies to Road, River and Mountain.

The details of how to encode FeatureCollection, FeatureMember and Feature are given by the GML2.0 specification, and are built in to the code generation tool presented later.

Since there is a different terminology at the ISO UML level and at the OGC XML level, a mapping must be identified. The mappings are done between elements that are semantically equivalent concepts in ISO and GML:

- The ISO basic data type *CharacterString* is mapped to the GML basic data type *siring*.
- The ISO basic data type *Date* is mapped to the GML basic data type *month*.
- The ISO class *GM_Curve* is mapped to the GML type *LineStringPropertyType*.

4. Two alternative code generation approaches

There are several possible approaches to make tools that generate code from UML. The output may be XML Schema as investigated in this paper, as well as other technologies like CORBA, DCOM, Java, C++ etc. This section describes two different approaches for developing code generation tools. To achieve UML tool independence, both approaches take XML Metadata Interchange (XMI) [14] as input. XMI at the UML model level is an OMG specification to describe the UML model in XML format, and most UML tools support XMI export.

4.1 COMDEF

COMDEF [8] (Figure 2) code generation tool is developed at SINTEF within several research projects (DISGIS, OBOE [9]). The first step is to export XMI from a UML tool. XMI is then transformed to a simpler and SINTEF proprietary format Component Modelling Language (CML) [8]. A specialised CML parser then reads the CML file and provides access to the CML elements through a Java API. To produce the final result, a Java application (Java Code Gen. in Figure 2) is implemented which accesses the CML Java API.

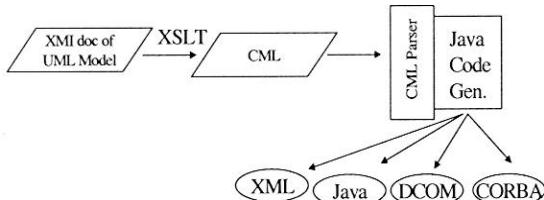


Figure 2 The process of COMDEF code generation tool

4.2 XSLT

XML Stylesheet Language Transformation (XSLT)[15] is a W3C specification for transforming XML into any text format, which for instance may be a single XML file or several Java files (Figure 3) depending on the wanted result. The only requirement is that the input must be an XML document, which makes XMI a proper input alternative. XSLT is a declarative language for describing the wanted transformations. An XSLT processor with built-in XML parser performs the transformations described by the XSLT document.

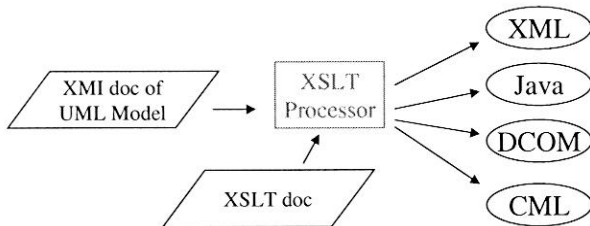


Figure 3 The process of using XSLT as a code generation tool

5. Architecture of the chosen code generation approach

Figure 4 shows the architecture of a code generation tool that generates GML2.0 from ISO conform UML models:

1. An ISO conformant application schema is defined in UML expressing the conceptual model at a detail level necessary for fully automatic code generation to GML2.0.
2. UML tools already support export to XMI.
3. XMI is an XML description of the UML model. The code generation tool takes XMI as input. Thus UML tool independence is achieved.

4. Due to the complexity of the XMI format, it is preferred to go from XMI to GML2.0 in a two-step process. First, XMI is transformed to a simpler XML format (simpleXMI).
5. simpleXMI expresses all the necessary information for code generation to GML2.0.
6. simpleXMI is transformed to GML2.0.
7. XML Schema according to GML2.0.

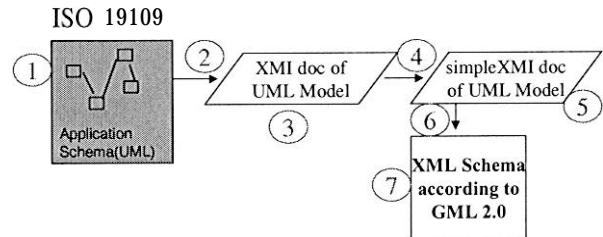


Figure 4 From UML to GML2.0

The intermediate format between XMI and the GML format is introduced to reduce the complexity of XMI. It is possible to generate GML from XMI directly, but then two complex tasks must be solved simultaneously:

- extracting the relevant model information from XMI
- applying the mapping rules from UML application schema to GML application schema

By introducing a new intermediate format, these two tasks can be nicely separated. By also making this intermediate format an XML format, it is possible to use XSLT for both transformations (items 4 and 6 of Figure 4). Thus an XSLT processor can be used in both cases, and there is no need to write a specialised parser. The XSLT processor takes care of the parsing and generation once proper XSLT files are defined. Two previously defined formats were considered as candidates for the intermediate format: Component Modelling Language (CML) and XML Config format, of ISO 19118 XML Encoding [6]. An evaluation of these formats is given in the next section.

6. Defining a simple format to express UML model information

XMI is a very complex format, with long XML tag names and lots of nested XML tags. The reasons for the complexity is that it is a generic format that applies to several meta levels within the OMG Meta Object Facility framework [12], current versions are DTD-based and not XML Schema-based, and it shall support a two-way mapping between XMI and a UML tool. Due to the complexity of the XMI format, it was chosen to use an intermediate, simpler format in the code generation process from XMI to GML. This intermediate format may be greatly simplified since it can be specialised for a one-way-mapping from UML models covering only the subset that is interesting for a code generation tool. Other code generation tools have used such formats with success. These are described and evaluated shortly along with a description of the resulting format that was used by the code generation tool described in this paper.

6.1 CML

This format is not XML-based: it is a plain text format that used OMG Interface Definition Language (IDL) as its starting point. The UML data types are mapped to IDL data types. Proper extensions were then added to CML to capture the necessary UML model info. This format has been used within the COMDEF code generation tool at SINTEF with several output formats, such as CORBA, XML, relational database schema, and Java. Thus it has been proven sufficiently expressive. To handle the CML format a separate parser must be written. This extra overhead can be avoided by defining the CML format as an XML format. XML formats have several standardised parsers and tools that can be used directly.

6.2 XML Config

XML Config is part of the ISO 19118 XML Encoding standard. This format is XML-based and an intermediate format between UML and the XML Encoding format. The major motivation for introducing this format in the ISO 19118 standard was to enable user configuration of the output XML encoding format. Thus there may be many possible XML formats from the same UML application schema. The drawback is the added complexity to support user configuration as well as putting at risk the interoperability. Two or more partners agreeing on the same UML application schema will not have the same XML format if different XML Config files are used.

XML Config is designed in order to support multiple inheritance in the UML model and removes all inheritance information by copying all attributes and relations from superclasses to corresponding subclasses. The reason for doing this is that XML Schema does not support multiple inheritance and flattening the inheritance structure is delegated to the scripts that produce XML Config. In addition it has several properties added for XML output configuration purposes only. This is not wanted in our simpleXMI format, since we want to have true UML descriptions only.

6.3 simpleXMI

The chosen format is given the name simpleXMI. It is a description of the UML model as with XMI, but it is simplified. The purpose of simpleXMI is to have enough information to do code generation, not to do a round-trip exchange between UML tools. simpleXMI is CML dressed up in XML syntax and without mapping of the UML data types. The UML information is kept unchanged so that code generation tools taking simpleXMI as input can decide on all the needed mappings itself. Currently simpleXMI meets all the needs of the city example and it will be further extended to meet all the needs of the real-life ISO application models yet to be defined in the GeNorway and NGIS project. The XML Schema in the next section, simpleXMI.xsd, defines simpleXMI.

6.3.1 simpleXMI.xsd

simpleXMI.xsd is the XML Schema that defines simpleXMI. <module> is the root element. The children of the <module>-element is <class>-elements. The children of a <class>-element consist of a list of <attribute>-elements followed by a list of <relationship>-elements. All of the elements have an attribute called name. <class> has two

additional attributes: **superclass** and **abstract**. <attribute> has one additional attribute: type. These attributes all map naturally to the UML concepts. <relationship> has four additional attributes, **otherClass**, **cardinality**, **collectionType** and **aggregationType**, that are explained shortly. **otherClass** is the name of the other class connected to this relationship. **cardinality** is the cardinality on the side of the other class. **collectionType** indicates the collection type for the other class side of relationships with cardinality larger than one. **collectionType** has two possible values: **set** or **sequence**. **aggregationType** describes the kind of aggregation containment of the other class. It has five possible values: **composite**, **aggregate**, **value**, **none**, and the empty string. **composite** means that UML composition is used, **aggregate** means that UML aggregation is used, and the three last ones are all equal to a UML association that is neither aggregation or composition.

simpleXMI document for the GML city example:

```
<module name="GMLCityExample"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="simpleXmi.xsd">
  <class name="Road" superClass="CityMember" abstract="false">
    <attribute name="classification" type="CharacterString"/>
    <attribute name="number" type="CharacterString"/>
    <attribute name="linearGeometry" type="GM_Curve"/>
  </class>
  <class name="River" superClass="CityMember" abstract="false">
    <attribute name="centerLineOf" type="GM_Curve"/>
  </class>
  <class name="Mountain" abstract="false">
    <attribute name="elevation" type="Integer"/>
  </class>
  <class name="CityModel" abstract="false">
    <attribute name="dateCreated" type="Date"/>
    <relationship name="cityMember" otherClass="CityMember"
      cardinality="0..*" collectionType="set"
      aggregationType="composite"/>
  </class>
  <class name="CityMember" abstract="true"/>
</module>
```

Extract from simpleXMI .xsd that validates the simpleXMI documents:

```
<xsd:element name="class" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="attribute" minOccurs="0"
        maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="name" type="xsd:string"/>
          <xsd:attribute name="type" type="xsd:string"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="relationship" minOccurs="0"
        maxOccurs="unbounded">
        ""
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="superClass" type="xsd:string"/>
    <xsd:attribute name="abstract" type="xsd:boolean"/>
  </xsd:complexType>
</xsd:element>
```

7. One UML model – many XML representations

One of the main advantages with being model-based is that the conceptual model represented by UML remains unchanged as new technical implementations are defined. The same conceptual model may be used to generate different XML Schemas such as GML2.0, ISO 19118 XML Encoding and XMI for the data level as well as other representations. Even if the world embraces a new technological infrastructure tomorrow, the conceptual UML model from yesterday remains the same. New mapping rules must be defined and new code generation tools must be implemented to support the new format. But this work is far more limiting than throwing away all the models and starting all over from scratch. This section will show different XML Schema output for the city UML conceptual model defined in section 2. To restrict the length of the examples, only the Road class is shown.

7.1 GML2.0 for the city UML model

The resulting GML2.0 Schema is the same as described in the GML2.0 specification, and it is derived by the code generation approach described in the previous chapters. The gml : -types are defined in the geometry.xsd and feature.xsd XML Schemas that are reused by all GML application schemas.

```
<element name="Road" type="RoadType"
  substitutionGroup="_CityMemberFeature"/>
<element name="_CityMemberFeature"
  type="gml:AbstractFeatureType" abstract="true"
  substitutionGroup="gml:_Feature"/>

<complexType name="RoadType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="classification" type="string"/>
        <element name="number" type="string"/>
        <element name="linearGeometry"
          type="gml:LineStringPropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

7.2 ISO 19118 XML Encoding for the city UML model

The resulting ISO 19118 XML Schema is derived by code generation tools that were partly developed within JNIP [2]. CharacterString, IM_ObjectIdentification and GM-Curve are defined within ISO/TC 211 standards-to-be. The ISO/TC 211 standards-to-be are all defined by UML models, and ISO 19118 XML Encoding specifies generic mapping rules that are used to derive corresponding XML Schemas.

```
<complexType name="Road">
  <sequence>
    <element name="classification" type="CharacterString"/>
    <element name="number" type="CharacterString"/>
    <element name="linearGeometry" type="GM_Curve"/>
  </sequence>
  <attributeGroup ref="IM_ObjectIdentification"/>
</complexType>
```

XMI also have mapping rules from UML to XMI for the data level, that is data according to instances of the classes described in the UML model. This must not be confused with XMI for the model level that has been discussed previously. By Using the XMI for the data level we may derive yet another XML Schema for the city UML conceptual model.

8. Conclusion and Future Work

OGC's GML specification plays a central role in OGC's successful Web Mapping Testbeds. ISO/TC 211 states that XML specifications should be derived from UML models using a model-driven architecture. SINTEF has tested the ISO/TC 211 principles on UML to GML generation. The code generation tool

demonstrates a bridge between ISO UML models and OGC XML implementations. The tool is valuable input to the ongoing harmonisation process of OGC and ISO/TC 211. There are several advantages with specifying the OGC GML application schemas by ISO UML models instead of directly in XML Schema:

- It is easier for users to specify the models using UML than to cope with all the technical details of an XML Schema
- It is easier to understand the models using UML
- It should reduce the number of errors in the GML schemas and reduce the time to develop such schemas.

The GeNorway project aims at developing the code generation tools further with real case models. GeNorway and the Norwegian Mapping Authority is just starting to define UML application models for all the feature catalogue types existing in the Norwegian proprietary SOSI standard. These UML models can then be automatically derived to international standards and specifications such as ISO 19118 XML Encoding, OGCs GML and OMGs XMI by defining mapping rules, and finally implementing the mapping rules by code generation tools. GML and ISO 19118 XML encoding are particularly designed for exchange of geodata, while XMI aims to be a cross-domain data exchange standard. SINTEF shall in the near future evaluate and compare these three different alternatives for exchange of geodata. It would be of great benefit if these three alternatives could be harmonised.

This paper describes how XSLT can be used as a code generation tool. The limitation is that XSLT can only be used when the input is XML. When generating from UML, XMI represents the XML format necessary. When the code generation result is XML, as described in this paper with GML, the XSLT code generation approach has several advantages compared to proprietary frameworks:

- There is no need to build specialised parsers. Off-the-shelf XSLT processors take care of the parsing and output generation.
- There is no need to install, buy and use language compilers.
- There is no need to maintain and understand proprietary classes and APIs.

It remains to investigate if XSLT is also preferred when non-XML output shall be generated.

9. References

- [1] Roy Grønmo, Arne-Jørgen Berre, Ida Solheim, Hjørdis Hoff and Kim Lantz, *DISGIS: An Interoperability Framework for GIS - Using the ISO/TC 211 Model-based Approach*. Global Spatial Data Infrastructure (GSDI) 4, Cape Town, South Africa. 2000.
- [2] Roy Grønmo and David Skogan, *SINTEF Report: Joint Nordic test case using ISO/TC 211 standards*, STF40 A01010. 2001.
- [3] ISO, *CD 19103 Geographic information - Part 3: Conceptual schema language*, ISO/TC 211 N 755. 1999.
- [4] ISO, *CD 19109.3 Geographic information - Rules for application schema*, ISO/TC 211 N 1009. November, 2000.
- [5] ISO, *CD 19118.2 Geographic information -Part 18. Encoding*, ISO/TC 211 N 917. May, 2000.
- [6] ISO, *ISO 19118 XML Based encoding rules*, September, 2000.
- [7] ISO, *ISO Technical Committee 211*: www.statkart.no/isotc211
- [8] Bård Kvalheim, *COMDEF a flexible framework for component development*. Cand.Scient, Departments of Informatics, University of Oslo. 1999.
- [9] OBOE, *Open Business Object Environment*: www.opengroup.org/oboe
- [10] OGC, *Geography Markup Language (GML) 2.0, OGC Recommendation Paper, 01-029*. February, 2001.
- [11] OGC, *Geography Markup Language (GML) 2.0, OGC Recommendation Paper -Listing 6.1 within Chapter 6.*, 01-029. February, 2001.
- [12] OMG, *Meta Object Facility (MOF) Specification, ad/97-08-14*. September 1, 1997.
- [13] OMG, *Model Driven Architecture*: www.omg.org/mda
- [14] OMG, *XML Metadata Interchange (XMI) Version 1.1*, OMG Document *ad/99-10-02*. October 25, 1999.
- [15] W3C, *Extensible Stylesheet Language Transformations*: www.w3.org/TR/xslt.html