Open Geospatial Consortium

Date: 2011-11-23

Reference number of this document: OGC 11-064r3r3

Category: Engineering Report

Editor: Clemens Portele, Reinhard Erstling

OGC® OWS-8 CCI Schema Automation Engineering Report

Preface

This document is a deliverable for the OGC Web Services 8 (OWS-8) testbed activity.

Document type:	OGC® Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

OWS testbeds are part of OGC's Interoperability Program, a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver proven candidate standards or revisions to existing standards into OGC's Standards Program, where they are formalized for public release. In OGC's Interoperability Initiatives, international teams of technology providers work together to solve specific geoprocessing interoperability problems posed by the Initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of OGC standards.

The OWS-8 sponsors are organizations seeking open standards for their interoperability requirements. After analyzing their requirements, the OGC Interoperability Team recommend to the sponsors that the content of the OWS-8 initiative be organized around the following threads:

* Observation Fusion

* Geosynchronization (Gsync)

* Cross-Community Interoperability (CCI)

* Aviation

More information about the OWS-8 testbed can be found at:

http://www.opengeospatial.org/standards/requests/74

OGC Document [11-139] "OWS-8 Summary Report" provides a summary of the OWS-8 testbed and is available for download:

https://portal.opengeospatial.org/files/?artifact_id=46176

Contents

Page

OGC® OWS-8 CCI Schema Automation Engineering Report

1 Introduction

1.1 Scope

This OGC® document specifies improvements to the processing of information represented in or referenced from an application schema in UML to create derived, implementation level resources, in particular:

- XML Schema documents to represent types and their properties
- Schematron schema documents to represent constraints
- XSLT-Stylesheets to create KML instances of features

The documented improvements have been specified, implemented in the ShapeChange tool and tested in the context of schemas developed as part of the NGA's Topographic Data Store (TDS) schemas.

The work is a continuation of the work documented in OGC® document 10-088r2, the OWS-7 Schema Automation Engineering Report.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Clemens Portele	interactive instruments
Reinhard Erstling	interactive instruments
Paul Birkel	Mitre

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2011-06-29	0.0.1	CP	all	First draft version
2011-08-17	0.0.2	RE	Clause 6, Annex A	
2011-09-01	0.1.0	CP	all	First release
2011-09-16	0.1.2	RE	Clause 6, Annex A	Changes in response to review by P.Birkel; examples
2011-09-21	0.2.0	CP		Updates after tests Examples

				changes in response to review by P.Birkel
2011-10-05	1.0.0	CP		Version for publication

1.4 Forward

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

Geography Markup Language, Version 3.2, Open Geospatial Consortium (OGC)

Geography Markup Language, Version 3.3 (draft), Open Geospatial Consortium (OGC)

ISO/TS 19103:2005, Geographic Information – Conceptual Schema Language

ISO 19109:2004, Geographic Information – Rules for Application Schemas

ISO 19115:2003, Geographic information – Metadata

ISO 19115:2003/Corr:2006, Geographic information – Metadata – Technical Corrigendum 1

ISO/IEC 19757-3:2006 Information technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron

KML, Version 2.2, Open Geospatial Consortium (OGC)

OMG Object Constraint Language, Version 2.2, OMG Document Number formal/2010-02-01

W3C XML Schema Part 1: Structures Second Edition. W3C Recommendation (28 October 2004)

W3C XML Schema Part 2: Datatypes Second Edition. W3C Recommendation (28 October 2004)

3 Terms and definitions

For the purposes of this report, the definitions specified in the documents listed in Clause 2 apply.

4 Conventions

4.1 Abbreviated terms

GML	Geography Markup Language
ISO	International Organization for Standardization
KML	formerly: Keyhole Markup Language
LTDS	Local Topographic Data Store[1]
MDR	U.S. DoD Metadata Registry[2]
NGA	National Geospatial-Intelligence Agency
OCL	Object Constraint Language
OGC	Open Geospatial Consortium
OWS	OGC Web Services
UML	Unified Modeling Language
WFS	Web Feature Service
XML	eXtended Markup Language
XPath	XML Path Language

4.2 UML notation

Diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram, as described in ISO/TS 19103.

[1] The Local Topographic Data Store (LTDS) specifies a set of feature types, geometries, attributes, and enumerants, as well as their individual specifications (e.g., definition, datatype, value range) and consists primarily of topographic features that are typically extracted and portrayed at 1:50K and 1:100K map/chart scales. See https://nsgreg.nga.mil/TopographicTerrestrial.jsp for additional information.

[2] The U.S. DoD Metadata Registry (MDR) supports the collection, storage and dissemination of structural metadata information resources (schemas, data elements, attributes, document type definitions, style-sheets, data structures etc.). This Web-based repository is designed to also act as a Clearinghouse through which industry and government coordination on metadata technology and related metadata issues can be advanced. The repository is accessible at https://metadata.ces.mil/mdr/.

5 OWS-8 schema automation overview

5.1 General remarks

This document specifies improvements to the processing of information represented in or referenced from an application schema in UML to create derived, implementation level resources. The following sections provide an overview of each target.

5.2 XML Schema documents

The XML Schema documents provide an XML realization of most of the information in the classifiers in the application schema. Encoding rules exist in GML 3.2 for application schemas and ISO/TS 19139 for metadata elements. The rules have already been extended to meet additional requirements.

In OWS-8, extensions focus on supporting the modeling of metadata profiles in UML and deriving appropriate XML Schema documents and Schematron schemas (see next sub-clause).

This topic is documented in Clause 6.

5.3 Schematron schema documents

In OWS-7 rules for the conversion of OCL to Schematron have been specified and implemented in ShapeChange. In OWS-8 this has been extended to cover additional OCL elements ("let") as well as additional encoding rules (GML 3.3, ISO/TS 19139).

This topic is documented in Clause 6.

5.4 XSLT-Stylesheets

In OWS-7, the KML encoding rule supported only a single KML style per feature type. In OWS-8 this has been extended to

- derive the styling of a feature in KML based on SLD/SE feature type styles;
- access the feature type styles as well as the kml:Style information from a portrayal registry

The transformation from GML 3.2/3.3 data to KML 2.2 data is specified as an XSLT-stylesheet. The resulting KML placemarks directly reference kml:Style provided by the portrayal registry.

This topic is documented in Clause 7.

6 XML Schema/Schematron encoding rule improvements

6.1 Metadata profiles

Metadata profiles are defined by means of UML classifiers, which are subtypes of the ISO 19115 classifiers. The following cases can be distinguished:

- Pure restrictions

  In this case the subtype (which may have the same local name as the supertype, but in the package of the profile schema) is only created to associate constraints with values of the supertype within the scope of the profile.

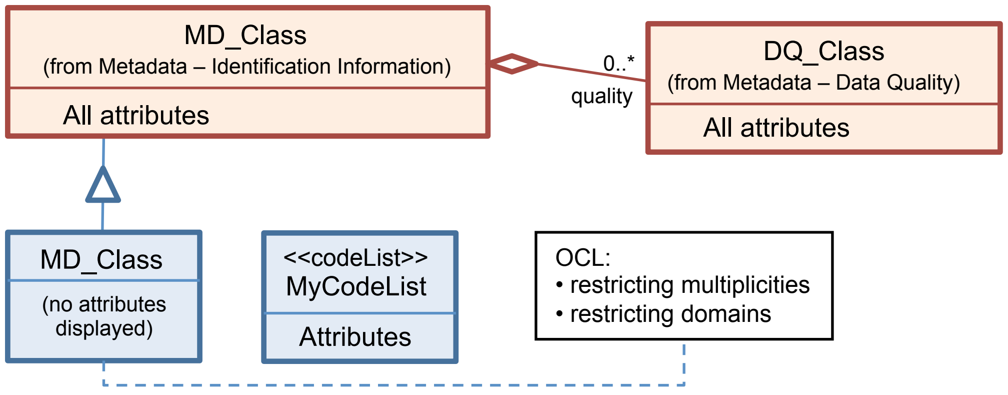

MD_Class
(from Metadata – Identification Information)
All attributes
0..*
quality
DQ_Class
(from Metadata – Data Quality)
All attributes
MD_Class
(no attributes displayed)
<<codeList>>
MyCodeList
Attributes
OCL:
• restricting multiplicities
• restricting domains

Figure 1 - Pure restriction case

Constraints are added as OCL constraints on properties of the supertype. Typical constraints are:

- Restrictions on the multiplicity of a property, including excluding a property from the profile, e.g., “property->isEmpty()”
- Restrictions on the domain of an attribute, including replacement of CharacterString by a code list, e.g., “attribute.oclIsTypeOf(MyCodeList)”
- Excluding code list values, e.g., “attribute <> CodeList::value”

UML/OCL compatibility considerations:

In ISO/TS 19103, code list values are not subtypes of CharacterString, so restricting the metadata properties that have a value type CharacterString to a code list is not strictly valid. However, since this is the method foreseen in ISO 19115, there are two possibilities. Both do not constitute sound solutions to the problem. They are mere workarounds for problems rooted in inconsistencies of the type definitions of key ISO standards, but may be justifiable because they are consistent with the rules for metadata profiles in ISO 19115; however, they will not work with general UML/OCL tools:

- Explicitly allow constraints as used above in the "pure restriction case", i.e. "attribute <> CodeList::value" and "attribute.oclIsTypeOf(MyCodeList)"

- Treat code lists as subtypes of CharacterString that limit the valid string values; i.e. explicitly allow constraints like "attribute <> 'value'" and "attribute.oclIsTypeOf(MyCodeList)"

Since code list values are more like enumerants than free strings, it was decided to follow the first of the two approaches in these encoding rule extensions.

The intended comparison operation

```
attribute<>CodeList::value
```

is a particularly problematic case, because equality operators such as “<>” and “=” in OCL are permitted between any objects of any type. The data types of the two comparands being complete unrelated (CharacterString vs. MyCodeList) the semantics of OCL would imply that the compared objects are different and the inequality operation (“<>”) of above expression would result in the constant *true*. Not surprisingly, implementing a far-reaching exception to the rules in the ShapeChange OCL system turned out to be difficult and dissatisfying. When the code generator translates a token such as the “attribute” operand, is usually does not know about the requirement that this in this special case is meant to be translated as if it were a code list of type CodeList.

In order to avoid deterioration of comprehensibility and clarity of the ShapeChange code is was decided to confine the rule conflict to the OCL type detection and type casting operations by requesting that in such comparisons as above an explicit type cast “oclAsType(CodeList)” has to be applied to the CharacterString operand. The comparison operation must therefore be written as follows:

```
attribute.oclAsType(CodeList)<>CodeList::value
```

Note that this solution is still invalid according to the strict rules of UML/OCL. The type cast operation above is supposed to deliver a result value of “null”, because the types are unrelated. The same is, of course, true for constructs employing “oclIsTypeOf(...)”.

XML encoding considerations:

In “pure restriction” case the subtype is just a container for the constraints and in general it is not intended to create a new object element for the subtype in the XML representation.

I.e, the representation in Figure 1 of the blue MD_Class in XML Schema is strictly not necessary as the Schematron rules encoding the restriction can capture all aspects of the classifier. However, in some cases it may be desirable to represent classes that do not add new properties in the XML Schema, too. An example might be that a community wants to express explicitly through the namespace of the metadata element that this is according to the profile associated with the namespace. I.e., we need a mechanism to control the behavior of the conversion. Following the guidance for encoding rule extensions planned for inclusion into GML 3.3, a tagged value is used for this: A tagged value *suppress* with a value of *'true'* attached to such pure restriction subtypes indicates that in the XML Schema no element, type and property type is to be created for this classifier.

Note that the use of this tagged value is only valid, if the classifier has a direct or indirect, non-abstract supertype without this tagged value set to 'true', and no subtype without this tagged value set to 'true'.

The meaning is that such a classifier does not appear in the XML Schema output (suppress!) and that any constraints defined for that classifier are being attached to the supertype.

Concerning the translation of OCL constraints to Schematron, the constraints found in a suppressed class are translated and then attached to its direct or indirect supertype, which is not suppressed. In this way you can import constraints into parts of the schema, which you otherwise cannot alter (such as ISO 19115 objects encoded by means of ISO/TS 19139).

There are two subtly different ways of thinking how the intended transfer of restrictions contained in a suppressed class to its superclass is supposed to work.

1. In a strict UML sense, the restricting constraints dwell in the suppressed class and only apply to instances of this class. In the mapping to XML schema the suppressed class is mapped to its superclass, which means that the XML instances of the superclass (and only these) are subject to the defined restrictions.
2. Another way of thinking is that the suppressed class is just a trick, a Trojan vehicle, to inject constraints into a class, which is otherwise unavailable to such an operation. In this case the constraints would be treated as if they were in reality specified in the superclass, which, of course, means that the “injected” constraints also apply to derivations of that superclass.

Since no decision about the intended behavior had been reached in the project both ways were implemented and made controllable via a ShapeChange configuration option. See section 6.4.1 for this.

Note that there are still some more unsettled questions beyond the strict-UML vs. Trojan interpretation concerning “suppress”.

What is the intended behavior, when the name of the suppressed class is used in OCL expressions? This can happen anywhere in the model, for example in a type constant used in oclIsTypeOf(…) or oclIsKindOf(…) or ….allInstances(). Is this being treated as if it *were* the superclass? Or should the strict UML rules apply?

Lacking answers to these questions, the current implementation just rejects class constants of suppressed classes in OCL expressions and emits appropriate error messages.

- Pure extensions

  In this case the subtype (which again may have the same local name as the supertype, but in the package of the profile schema) represents an intent to revise the behavior of the supertype purely by additional properties.

  New attributes, roles, and classifiers representing domains of new properties are added.

  Inherited properties are used "as is" without any constraints on them.

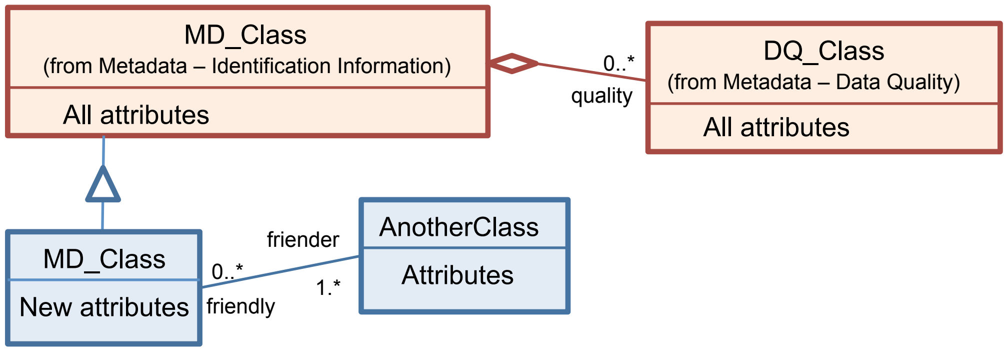


Figure 2 - Pure extension case

- Combinations of restrictions and extensions

  This case represents a combination of the previous two cases. Additional properties are added to the subtype in the profile as well as constraints.

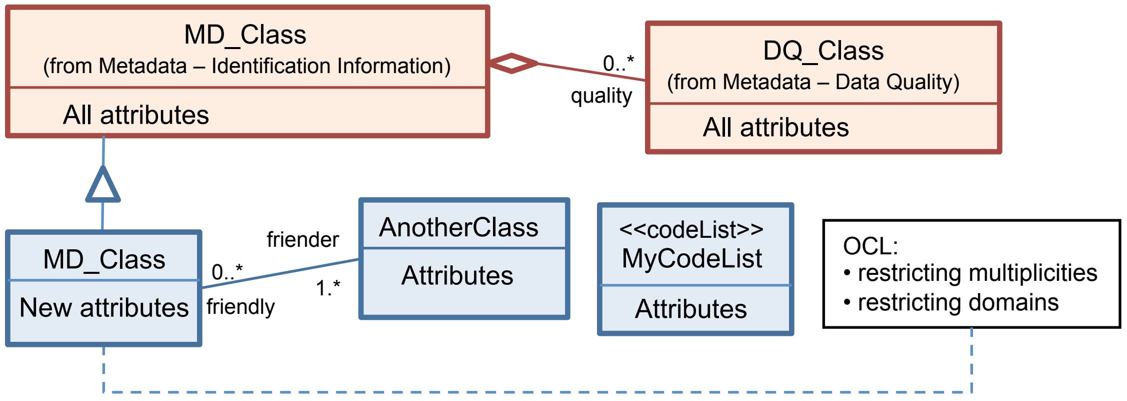


Figure 3 – Combined case

To enable a consistent encoding in XML, the encoding rule of the subtype and its sub-elements as specified in the tagged value *xsdEncodingRule* shall be the same as in the supertype. This applies to all types in an application schema.

6.2 UML profile extensions for code lists and units of measurements

In order to enable constraints on code list values and units of measurements that cannot be properly encoded using OCL, additional tagged values are specified for these cases. The tagged values are specified in the following tables.

Table 1 – Tagged values for code list value constraints

Stereotype	Model element	Tagged Value	Description
codeList	classifier	*codeList*	Base URI of the code list
		codeListValuePattern	Value access pattern for the code list, containing the substitution points {codeList} and {value}, where {codeList} is the base URI of the code list (replaced by the tagged value codeList) and {value} the local identifier of the code list value. Default: "{codeList}/{value}"
		codeListRepresentation	MIME type indicating the code list representation Valid values are: "application/gml+xml;version=3.2" for a GML dictionary (as currently used in the MDR) "application/rdf+xml" for a SKOS concept scheme (as created as part of the LTDS RDF representation) Default: "application/gml+xml;version=3.2"

Table 2 – Tagged values for unit of measure value constraints

Stereotype	Model element	Tagged Value	Description
schema or application Schema	Package	*uomResourceURI*	Base URI of the units dictionary Example: "http://metadata.ces.mil/mdr/ns/GSIP/uom/"
		uomResourceValuePattern	Access pattern for the unit, containing the following substitution points: • {resource}: The base URI of the units dictionary, to be replaced by the tagged value uomResourceURI from the schema. • {quantity}: The quantity type of the unit, to be replaced with the value of the tagged value physicalQuantity (or the value 'noncomparable') from the property. • {uom}: The local identifier of the unit in the units dictionary, to be replaced with the value of the tagged value recommendedMeasure or noncomparableMeasure from the property, or a valid value from the resource that represents the physicalQuantity. Default: "{resource}/{quantity}/{uom}".
		uomResourceRepresentation	MIME type indicating the units dictionary representation Currently, only one value is specified: "application/gml+xml;version=3.2" for a GML dictionary (as currently used in the MDR) Default: "application/gml+xml;version=3.2"
-	attribute	*physicalQuantity*	Physical quantity of the referenced unit Example: "length"
		recommendedMeasure	Unit recommended for use with this property. The unit must be consistent with the physicalQuantity value. Example: "metre"
		noncomparableMeasure	Valid non-comparable unit. Example: "flightLevel"

6.3 XML Schema conversion rule extensions implemented in ShapeChange

6.3.1 Example model

As an example we use the following UML model of a profile:

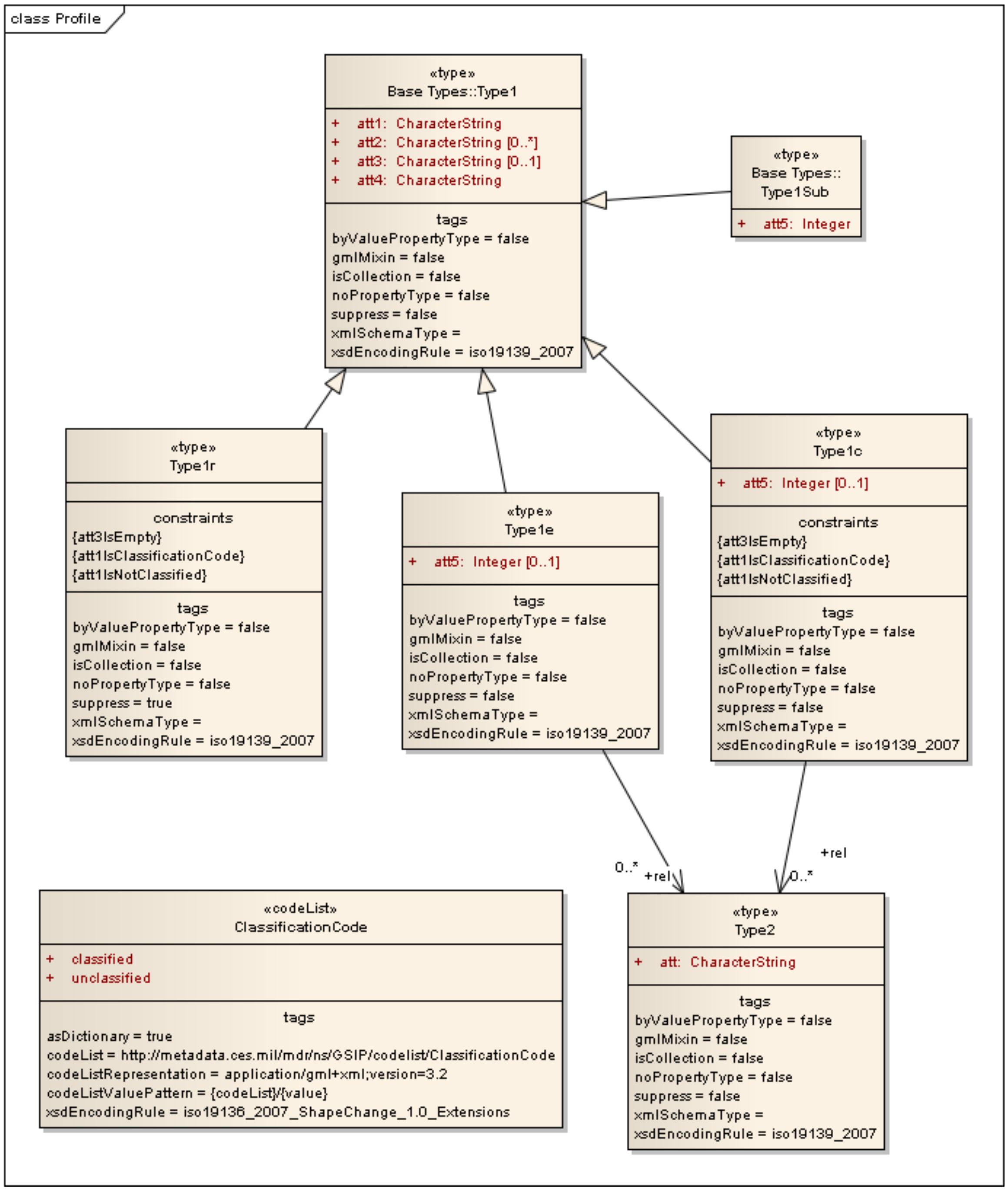


Figure 4 - Test model for ISO 19115 profiles

6.3.2 Tagged value suppress

A classifier that has the tagged value *suppress* with a value of *'true'* shall not be represented in the XML Schema, i.e. no object element, no type and no property type is created for this classifier.

If this classifier is used as a value of a property, the property type of the direct or indirect, non-abstract supertype without this tagged value set to *'true'* shall be used as the property type of this type.

In Figure 4 the type Type1r (restriction) is 'suppressed' and thus no XML schema component for this type is defined as part of the profile schema, while the standard XML schema components are created for Type1e (extension) and Type1c (combination). This shows the schema components created for Type1c:

```
<element name="Type1c" substitutionGroup="b:Type1"
 type="p:Type1c_Type"/>

<complexType name="Type1c_Type">
  <complexContent>
    <extension base="b:Type1_Type">
      <sequence>
        <element minOccurs="0" name="att5"
         type="gco:Integer_PropertyType"/>
        <element maxOccurs="unbounded" minOccurs="0" name="rel"
         type="p:Type2_PropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="Type1c_PropertyType">
  <sequence minOccurs="0">
    <element ref="p:Type1c"/>
  </sequence>
  <attributeGroup ref="gco:ObjectReference"/>
  <attribute ref="gco:nilReason"/>
</complexType>
```

6.3.3 XML Schema encoding of code list properties of features

There are two options how to represent code list values in GML. One is the default GML 3.2 encoding of code list values (gml:CodeType), the other is to already use the new recommended encoding with GML 3.3 (gml:ReferenceType, i.e., an xlink:href to the value).

In the ShapeChange implementation the parameter indicating the target GML version determines the code list encoding rule applied.

For target version "3.2", a feature attribute "att1" that is code-list-valued is converted to

```
<element name="att1" type="gml:CodeType"/>
```

while for target version "3.3", the same feature attribute is converted to

```
<element name="att1" type="gml:ReferenceType"/>
```

6.3.4 Association classes

The GML 3.3 association class encoding rule has been implemented in ShapeChange.

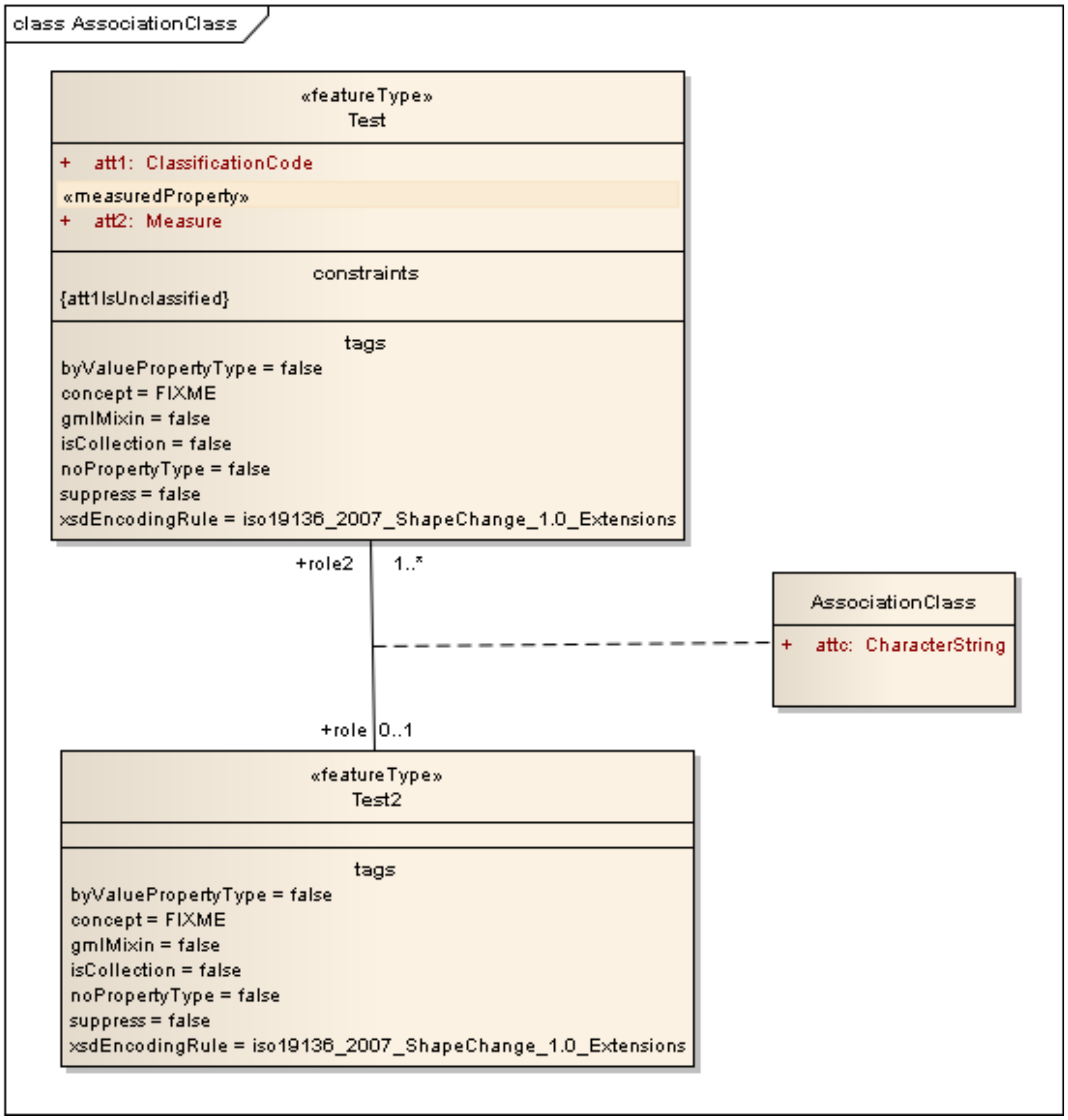


Figure 5 – Association class example

The model shown in the figure above is converted in XML Schema to:

```
<element name="AssociationClass"
  substitutionGroup="gml:AbstractGML"
  type="t:AssociationClassType"/>
```

```
<complexType name="AssociationClassType">
  <complexContent>
    <extension base="gml:AbstractGMLType">
      <sequence>
        <element name="attc" type="string"/>
        <element name="role" type="t:Test2PropertyType"/>
        <element name="role2" type="t:TestPropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="AssociationClassPropertyType">
  <sequence minOccurs="0">
    <element ref="t:AssociationClass"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>

<element name="Test" substitutionGroup="gml:AbstractFeature"
 type="t:TestType"/>

<complexType name="TestType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="att2" type="gml:MeasureType"/>
        <element name="att1" type="gml:ReferenceType"/>
        <element minOccurs="0" name="role"
         type="t:AssociationClassPropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="TestPropertyType">
  <sequence minOccurs="0">
    <element ref="t:Test"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>

<element name="Test2" substitutionGroup="gml:AbstractFeature"
 type="t:Test2Type"/>

<complexType name="Test2Type">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element maxOccurs="unbounded" name="role2"
         type="t:AssociationClassPropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="Test2PropertyType">
  <sequence minOccurs="0">
    <element ref="t:Test2"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
```

6.4 Schematron conversion rule extensions implemented in ShapeChange

6.4.1 Tagged value suppress

As explained in section 6.1 there are two options, which control, how the transfer of OCL content to the supertype is performed.

You choose one of these options by means of the ShapeChange *targetParameter* named *suppressedTypeInterpretation* by selecting one of the values *strictUML* or *trojanType* as follows:

```
<targetParameter name="suppressedTypeInterpretation" value="strictUML"/>
```

or

```
<targetParameter name="suppressedTypeInterpretation" value="trojanType"/>
```

Default value is *strictUML*.

The meaning of the options is as follows:

- *strictUML*: Using this option only the direct or indirect superclass (which also must be non-abstract) will receive the Schematron assertions, which are translated from the OCL constraints.

- *trojanType*: With this option the Schematron assertions are attached to the non-suppressed superclass as if the constraints were actually denoted there. This means that the class itself (if it is concrete) and all concrete derived subclasses will receive the assertions.

As an example consider the UML model in Figure 4.

All types in the model depicted in Figure 4 are tagged with *xsdEncodingRule iso19139_2007*. Class Type1 and derived type Type1Sub are supposed to belong to the base types in the metadata model.

Type1r is the only class with tagged value *suppress='true'*.

Result with *suppressedTypeInterpretation=strictUML*:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
 xmlns:sch="http://purl.oclc.org/dsdl/schematron">
 <title>Schematron constraints for schema 'Profile'</title>
 <ns prefix="sch" uri="http://purl.oclc.org/dsdl/schematron"/>
 <ns prefix="p" uri="http://www.opengis.net/ows8/test/1"/>
 <ns prefix="b" uri="http://www.opengis.net/ows8/test/3"/>
 <pattern>
  <rule context="b:Type1">
   <let name="A" value="b:att1/*[name()='p:ClassificationCode']"/>
   <assert test="b:att1/*[name()='p:ClassificationCode']">att1IsClassificationCode: att1 is
    restricted to ClassificationCode values </assert>
   <assert
    test="concat($A/@codeList,'/',$A/@codeListValue) !=
'http://metadata.ces.mil/mdr/ns/GSIP/codelist/ClassificationCode/classified'"
    >att1IsNotClassified: att1 is not of value classified </assert>
   <assert test="not(b:att3/*)">att3IsEmpty: att3 not part of the profile </assert>
  </rule>
  <rule context="p:Type1c">
   <let name="A" value="b:att1/*[name()='p:ClassificationCode']"/>
   <assert test="b:att1/*[name()='p:ClassificationCode']">att1IsClassificationCode: att1 is
    restricted to ClassificationCode values </assert>
   <assert
    test="concat($A/@codeList,'/',$A/@codeListValue) !=
'http://metadata.ces.mil/mdr/ns/GSIP/codelist/ClassificationCode/classified'"
    >att1IsNotClassified: att1 is not of value classified </assert>
   <assert test="not(b:att3/*)">att3IsEmpty: att3 not part of the profile </assert>
  </rule>
 </pattern>
</schema>
```

Note that there is no Schematron rule for the suppressed class Type1r.

Its constraints appear as Schematron assertions for objects of type Type1 (mapped to element *b:Type1*) belonging to the base model.

Result with *suppressedTypeInterpretation=trojanType*:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
 xmlns:sch="http://purl.oclc.org/dsdl/schematron">
 <title>Schematron constraints for schema 'Profile'</title>
 <ns prefix="sch" uri="http://purl.oclc.org/dsdl/schematron"/>
 <ns prefix="p" uri="http://www.opengis.net/ows8/test/1"/>
 <ns prefix="b" uri="http://www.opengis.net/ows8/test/3"/>
 <pattern>
  <rule context="b:Type1">
   <let name="A" value="b:att1/*[name()='p:ClassificationCode']"/>
   <assert test="b:att1/*[name()='p:ClassificationCode']">att1IsClassificationCode: att1 is
    restricted to ClassificationCode values </assert>
   <assert
    test="concat($A/@codeList,'/',$A/@codeListValue) !=
'http://metadata.ces.mil/mdr/ns/GSIP/codelist/ClassificationCode/classified'"
    >att1IsNotClassified: att1 is not of value classified </assert>
   <assert test="not(b:att3/*)">att3IsEmpty: att3 not part of the profile </assert>
  </rule>
  <rule context="b:Type1Sub">
   <let name="A" value="b:att1/*[name()='p:ClassificationCode']"/>
   <assert test="b:att1/*[name()='p:ClassificationCode']">att1IsClassificationCode: att1 is
```

```
     restricted to ClassificationCode values </assert>
    <assert
     test="concat($A/@codeList,'/',$A/@codeListValue) !=
'http://metadata.ces.mil/mdr/ns/GSIP/codelist/ClassificationCode/classified'"
     >att1IsNotClassified: att1 is not of value classified </assert>
    <assert test="not(b:att3/*)">att3IsEmpty: att3 not part of the profile </assert>
   </rule>
   <rule context="p:Type1e">
    <let name="A" value="b:att1/*[name()='p:ClassificationCode']"/>
    <assert test="b:att1/*[name()='p:ClassificationCode']">att1IsClassificationCode: att1 is
     restricted to ClassificationCode values </assert>
    <assert
     test="concat($A/@codeList,'/',$A/@codeListValue) !=
'http://metadata.ces.mil/mdr/ns/GSIP/codelist/ClassificationCode/classified'"
     >att1IsNotClassified: att1 is not of value classified </assert>
    <assert test="not(b:att3/*)">att3IsEmpty: att3 not part of the profile </assert>
   </rule>
   <rule context="p:Type1c">
    <let name="A" value="b:att1/*[name()='p:ClassificationCode']"/>
    <assert test="b:att1/*[name()='p:ClassificationCode']">att1IsClassificationCode: att1 is
     restricted to ClassificationCode values </assert>
    <assert
     test="concat($A/@codeList,'/',$A/@codeListValue) !=
'http://metadata.ces.mil/mdr/ns/GSIP/codelist/ClassificationCode/classified'"
     >att1IsNotClassified: att1 is not of value classified </assert>
    <assert test="not(b:att3/*)">att3IsEmpty: att3 not part of the profile </assert>
   </rule>
  </pattern>
</schema>
```

In this case all classes deriving from Type1 obtain the constraints specified in Type1r, which is, of course, excluded, because it is suppressed. The effect is the same as if the constraints were indeed specified in the class Type1 itself.

Note that also Type1e and Type1c receive the constraints by way of inheritance. However, those in Type1c are overwritten with ones of identical names defined in the class itself.

6.4.2 Schematron encoding of OCL constraints in metadata profiles

For those parts of the UML model, which represent metadata, the encoding rules of ISO/TS 19139 apply (indicated by a tagged value xsdEncodingRule with a value of 'iso19139_2007'). These encoding rules are similar to the GML encoding rules, but different in some details. If metadata properties are addressed in OCL constraints, these different encoding rules have to be taken into account.

The following kinds of properties have to be encoded differently in Xpath expressions:

- Properties where the value is simple

  When translating OCL syntax according to the GML encoding rules, properties carrying simple types are mapped to an XPath expression ending in the name of the property. However, simple type properties encoded according to ISO/TS 19139 embed an additional element, which encodes the property type. This additional element has to be accounted for, which is done by appending an additional */** to the property name path as in:

metadatastring/*

- Properties where the value is a code list or enumeration

  Code lists and enumerations in GML (now disregarding the changes concerning code lists which are being made for GML 3.3) are translated like simple types, effectively they are translated as if they were character strings. In ISO/TS 19139 this is also the case for enumerations. However, the encoding of code lists is different. The values of code lists are carried on an attribute named *codeListValue*, which resides on an embedded element substituted for the element <CodeListValue>. So, ISO/TS 19139 code list properties have to be translated by appending */*/@codeListValue* to the property name path, such as:

  metadatacodelist/*/@codeListValue

- Properties with nil values

  Nil valued properties in GML encoding carry the *xsi:nil* attribute and, optionally, also the *nilReason* attribute. The access to the reason value is therefore translated to the property path with an additional phrase [@xsi:nil='true']/@nilReason appended. In ISO/TS 19139 encoding *xsi:nil* is not employed. Instead of that the content of the property is omitted and the property carries a *gco:nilReason* attribute. Therefore, for ISO/TS 19139 access to the reason value must be translated to the property path extended by an additional phrase [not(*)]/@gco:nilReason:

  metadataproperty[not(*)]/@gco:nilReason

6.4.3 Schematron encoding of OCL constraints on code lists in GML 3.3

When using the GML 3.3 target conventions, code-list-typed properties used in OCL comparison operations are translated under the influence of several tagged values, which are attached to the code list class.

The following tagged values apply (see Table 1):

- *codeList* (mandatory)
- *codeListValuePattern* (optional, default is: "{codeList}/{value}")

An OCL comparison involving a code list property, such as

codelistproperty <> CodeListClass::codelistvalue

is translated as follows:

The property part *codelistproperty* is translated to the value of the *xlink:href* attribute:

codelistproperty/@xlink:href

The code list constant *CodeListClass::codelistvalue* is translated to the substituted *codeListValuePattern* (see table above). The tagged value for *codeList* is substituted for {codeList} and the code list value ("*codelistvalue*") is inserted for {value}.

If the *codeList* tag on the class *CodeListClass* has the value *http://server/CodeListClass*, we get

http://server/CodeListClass/codelistvalue

for the XPath equivalent of the code list constant.

So in the end the comparison will result in:

codelistproperty/@xlink:href != 'http://server/CodeListClass/codelistvalue'

6.4.4 Support for OCL "let ... in"

The OCL construct

let *variable : type = expression, ...* ***in*** *expression-using-variable(s)*

permits to define and initialize variables and subsequently make use of these in an embedded expression. The idea of *let* is a short-hand notation for expressions, which contain more than one occurrences of a sub-expression. More than one variable can be defined and initialized in one *let* expression. The *type* (including the preceding colon (:) can be omitted. In this case the type is derived from the initializing expression.

Translating *let* expressions in the OCL-to-Schematron context had not been done in OWS-7, because it was not deemed a necessary construct. Everything, which can possibly be expressed using a *let*, can also be expressed without. However, the use of OCL constraints sometimes produces quite lengthy constraints, the readability of which benefits a lot from providing *let*.

Therefore the following approach (originally described in the OWS-7 Schema Automation ER) has been implemented:

- *let* expressions at the outmost level of expressions are translated to the Schematon <let> element. The <let> element definitions are evaluated by Schematron for each object in the <rule> context, the <let> is residing in. Therefore only those parts of the OCL expression, which translate in the context of *current()* may be treated in this way.

- *let* expressions in deeper nesting levels, where the variable initialization explicitly or implicitly refers to bound variables of iterators, require substitution of the translated expression in all places, where the defined variable is used.

The implementation of *let* is quite general. The *let* construct can be written anywhere in an OCL expression, where an arbitrary OCL sub-expression is permitted. Exceptions to this rule are those places in the syntax, which in the present implementation already had to endure restrictions on OCL expression syntax due to the limitations of XPath 1. A prominent example for such a restriction: The body of the *unique()* iterator. In these places still only the restricted syntax is admissible and neither *let* nor a *let*-defined variable can be employed.

A small example will illustrate the translation of the "let … in" construct:

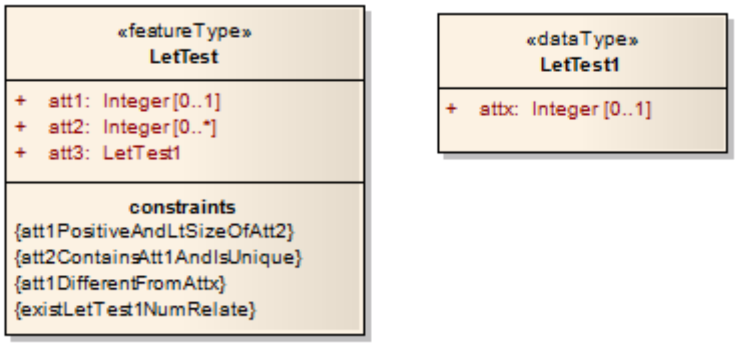


Figure 6 - Model for testing "let"

Class LetTest contains the following constraints:

```
/* att1 must be greater 0 and less than the cardinality of att2. */
inv att1PositiveAndLtSizeOfAtt2: let u=att1, v=att2 in u->notEmpty() implies 0 < u and u < v->size()

/* att2 contains att1 and is unique. */
inv att2ContainsAtt1AndIsUnique: let x=att1,y=att2 in x->notEmpty() implies y->exists(z|z=x) and y->isUnique(z|z)

/* att1 must be different from associated attx.*/
inv att1DifferentFromAttx: let a=att1, b=att3.attx in a->notEmpty() and b->notEmpty() implies a<>b

/* There need to exist LetTest1 objects, which numerically relate to the current object.*/
inv existLetTest1NumRelate: let a=att1 in LetTest1.allInstances()->exists( x | let b=x.attx in 0<=b and b<2*a )
```

The generated Schematron code for these two classes and the four constraints above are as follows:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
 xmlns:sch="http://purl.oclc.org/dsdl/schematron">
 <title>Schematron constraints for schema 'Test'</title>
 <ns prefix="sch" uri="http://purl.oclc.org/dsdl/schematron"/>
 <ns prefix="t" uri="http://www.opengis.net/ows8/test/2"/>
 <pattern>
  <rule context="t:LetTest">
   <let name="A" value="t:att1"/>
   <let name="B" value="t:att3/*/t:attx"/>
   <let name="C" value="t:att2"/>
   <assert test="not($A) or not($B) or $A != $B">att1DifferentFromAttx: att1 must be different
    from associated attx.</assert>
   <assert test="not($A) or 0 &lt; $A and $A &lt; count($C)">att1PositiveAndLtSizeOfAtt2:
    att1 must be greater 0 and less than the cardinality of att2. </assert>
   <assert
    test="not($A) or $C[. = $A] and not($C[. = (preceding::*|ancestor::*)[count(.|$C)=count($C)]])"
    >att2ContainsAtt1AndIsUnique: att2 contains att1 and is unique. </assert>
   <assert test="(//t:LetTest1)[0 &lt;= t:attx and t:attx &lt; 2 * $A]"
    >existLetTest1NumRelate: There need to exist LetTest1 objects, which numerically relate to
    the current object.</assert>
  </rule>
 </pattern>
</schema>
```

let variable names are generally not preserved. They cannot be preserved because even in one single OCL constraint the same variable name can be employed in distinct *let* … *in* usages, each spanning a different scope.

The implementation collects all Schematron assertions for one feature type in one single Schematron rule (see 6.4.5 for an explanation of this behavior). When doing this, all generated <let> variable names and values are kept track of, identifying identical expressions.

Uses of *let* … *in* referring to variables bound to iterators cannot be translated to <let> elements, because <let> elements are executed in the context of *current()*, which stands for the object, generally the feature type treated by a Schematron rule. The last one of the examples shows this. Here x.attx needs to be substituted for each occurrence of b.

If *let*… were to be translated for a Schematron environment with XPath 2 support, it would be possible to avoid substitution of *let* variables defined in deeper nesting levels. Though XPath 2 still does not provide for a direct equivalent of an assignment, the XPath 2 *for expression* might be utilized to provide a similar effect.

6.4.5 Schematron document structure improvements

Experience from other project work (outside of the OWS initiatives) had shown that the nesting of the Schematron elements <pattern>, <rule> and <assert> had not been done in an optimal way in OWS-7. In response to this insight the change described in this section had been established. The change is therefore not part of the OWS-8 activity, but has been contributed additionally and in-kind. It is described here, because it constitutes a rather large and visible intervention into the formerly known ShapeChange program behavior.

In OWS-7 a generated Schematron schema consisted of one (big) <pattern> element, which contained one <rule> for each translated OCL constraint. Consequently each <rule> element contained exactly one <assert>, which carried the result of the translation of the OCL constraint.

The following Schematron schema snippet from OWS-7 testing material exemplifies this kind of structure.

```
 <pattern>
   <rule context="nas:AircraftHangar">
    <assert test="nas:address/*">AddressReq: Building</assert>
   </rule>
   <rule context="nas:AircraftHangar">
    <assert
      test="not(nas:featureFunction/*/nas:valuesOrReason) or not((nas:featureFunction/*/nas:valuesOrReason)[. != 
'aircraftRepair' and . != 'airTransport' and . != 'cargoHandling' and . != 'diningHall' and . != 'dormitory' and . != 
'emergencyOperations' and . != 'emergencyShelter' and . != 'meetingPlace' and . != 'outPatientCare' and . != 
'warehousingStorage']) "
      >FeatureFunction: AircraftHangar</assert>
   </rule>
   <rule context="nas:AircraftHangar">
    <assert
      test="not(nas:verticalConstMaterial/*/nas:valuesOrReason) or 
not((nas:verticalConstMaterial/*/nas:valuesOrReason)[. != 'adobeBrick' and . != 'brick' and . != 'concrete' and . != 
'masonry' and . != 'metal' and . != 'prestressedConcrete' and . != 'reinforcedConcrete' and . != 'steel' and . != 'wood']) "
      >VertConstMaterial: AircraftHangar</assert>
   </rule>
   <rule context="nas:AircraftHangar">
    <assert test="nas:verticalConstMaterial/*">VertConstMaterialReq: AircraftHangar</assert>
   </rule>
   ….
 </pattern>
```

As it turned out this simple structure is not optimal, because the rules of Schematron demand that the <rule>s belonging to one <pattern> are treated in an "if … then … else if … else if …" fashion.

In the Schematron standard you find the statements:

6.5 Order and side-effects
…
The only elements for which order is significant are the `rule` and `let` elements.
A `rule` element acts as an if-then-else statement within each pattern. An implementation may make order non-significant by converting rules context expressions to elaborated rule context expressions[3].

This means that in executing a <pattern> on a single element instance, the instance bubbles down the <rule>s of the <pattern>, until it finds a match on the context. The <assert>s in this <rule> are executed and the rest of the remaining <rule>s are discarded.

In the way <pattern>s were structured in OWS-7, this means that for one particular feature type (nas:AircraftHangar in the example), only the first <rule> would apply, which is quite unfortunate.

Therefore, the Schematron <pattern> structure has been changed and has been made available to OWS-8. In the new structure there is only one <rule> for each feature type

[3] An *elaborated rule context expression* is one, which does not match any of the context expressions in other rules. The "if then else if" logic does not apply to such elaborated contexts because the all match something different.

and all <assert>s (corresponding to the OCL constraints) belonging to that feature type are now gathered in one <rule>.

The following Schematron snippet shows the difference:

```
<pattern>
  <rule context="nas:AircraftHangar">
    <assert test="nas:address/*">AddressReq: Building</assert>
    <assert
      test="not(nas:featureFunction/*/nas:valuesOrReason) or not((nas:featureFunction/*/nas:valuesOrReason)[. !=
'aircraftRepair' and . != 'airTransport' and . != 'cargoHandling' and . != 'diningHall' and . != 'dormitory' and . !=
'emergencyOperations' and . != 'emergencyShelter' and . != 'meetingPlace' and . != 'outPatientCare' and . !=
'warehousingStorage']) "
      >FeatureFunction: AircraftHangar</assert>
    <assert
      test="not(nas:verticalConstMaterial/*/nas:valuesOrReason) or
not((nas:verticalConstMaterial/*/nas:valuesOrReason)[. != 'adobeBrick' and . != 'brick' and . != 'concrete' and . !=
'masonry' and . != 'metal' and . != 'prestressedConcrete' and . != 'reinforcedConcrete' and . != 'steel' and . != 'wood'])"
      >VertConstMaterial: AircraftHangar</assert>
    <assert test="nas:verticalConstMaterial/*">VertConstMaterialReq: AircraftHangar</assert>
  </rule>
  ….
</pattern>
```

In the new document structure all Schematron assertions derived from OCL constraints for one feature type are applied to each feature instance. This is in contrast to the former behavior where only the first rule in the pattern was effective.

6.4.6 Additional assertions on code list values and constraints

6.4.6.1 Target encoding: GML 3.3

If the code list classifier has a tagged value codeList ({codeList}) then the following assertion is added to the Schematron schema in the context of the property:

starts-with(./@xlink:href,{codeList})

To test the existence of the code list value the following assertion is added to the Schematron schema in the context of the property:

(not contains(@xlink:href, '#') and document(./@xlink:href)) or

(contains(@xlink:href, '#') and document(substring-before(./@xlink:href,'#'))/id(substring-after(./@xlink:href,'#')))

In addition, we can assert that the remote resource has the correct element based on its representation.

For "application/gml+xml;version=3.2" (the default) we expect a gml:Definition:

(not contains(@xlink:href, '#') and document(./@xlink:href)/gml:Definition) or (contains(@xlink:href, '#') and document(substring-before(./@xlink:href,'#'))/id(substring-after(./@xlink:href,'#'))[local-name()='Definiton' and namespace-uri()='http://www.opengis.net/gml/3.2'])

For "application/rdf+xml" we expect a skos:Concept (see the OWS-8 Semantic Mediation ER):

(not contains(@xlink:href, '#') and document(./@xlink:href)/skos:Concept) or

(contains(@xlink:href, '#') and document(substring-before(./@xlink:href,'#'))/id(substring-after(./@xlink:href,'#'))[local-name()='Concept' and namespace-uri()='http://www.w3.org/2004/02/skos/core#'])

For example, for the property Test.att1 shown in Figure 5 the following Schematron assertions are created:

```
    <rule context="t:Test">
      <assert test="starts-
with(t:att1/@xlink:href,'http://metadata.ces.mil/mdr/ns/GSIP/code
list/ClassificationCode')">Code list value URI starts with
'http://metadata.ces.mil/mdr/ns/GSIP/codelist/ClassificationCode'
</assert>
      <assert test="(not contains(t:att1/@xlink:href, '#') and
document(t:att1/@xlink:href)) or (contains(t:att1/@xlink:href,
'#') and document(substring-
before(t:att1/@xlink:href,'#'))/id(substring-
after(t:att1/@xlink:href,'#')))">Code list value exists</assert>
      <assert test="(not contains(t:att1/@xlink:href, '#') and
document(t:att1/@xlink:href)/gml:Definition) or
(contains(t:att1/@xlink:href, '#') and document(substring-
before(t:att1/@xlink:href,'#'))/id(substring-
after(t:att1/@xlink:href,'#'))[local-name()='Definiton' and
namespace-uri()='http://www.opengis.net/gml/3.2'])">Code list
dictionary is represented using GML 3.2</assert>
    </rule>
```

6.4.6.2 Target encoding: GML 3.2

For GML 3.2, the expressions are different as the code list information is split into the codeSpace attribute and the text node.

To verify the codeSpace attribute, the following assertion is added to the Schematron schema in the context of the property, if the tagged value {codeList} has been provided:

@codeSpace={codeList}

To test the existence of the code list value the following assertion is added to the Schematron schema in the context of the property. In {codeListValuePattern} we replace "{codeList}" by {codeList} and "{value}" by "*":

(not contains('{codeListValuePattern}', '#') and document('{codeListValuePattern}')) or (contains('{codeListValuePattern}', '#') and document(substring-before('{codeListValuePattern}','#'))/id(substring-after('{codeListValuePattern}','#')))

In addition, we can assert that the remote resource has the correct element based on its representation.

For "application/gml+xml;version=3.2" (the default) we expect a gml:Definition:

(not contains('{codeListValuePattern}', '#') and document('{codeListValuePattern}') /gml:Definition) or (contains('{codeListValuePattern}', '#') and document(substring-before('{codeListValuePattern}','#'))/id(substring-after('{codeListValuePattern}','#')) [local-name()='Definiton' and namespace-uri()='http://www.opengis.net/gml/3.2'])

For "application/rdf+xml" we expect a skos:Concept (see the OWS-8 Semantic Mediation ER):

(not contains('{codeListValuePattern}', '#') and document('{codeListValuePattern}') /skos:Concept) or (contains('{codeListValuePattern}', '#') and document(substring-before('{codeListValuePattern}','#'))/id(substring-after('{codeListValuePattern}','#')) [local-name()='Concept' and namespace-uri()=' http://www.w3.org/2004/02/skos/core#'])

6.4.6.3 Target encding: ISO/TS 19139

The Schematron assertions for code list values in properties encoded according to ISO/TS 19139 is similar to the GML 3.2 encoding. The differences are described in 6.4.2.

To verify the codeSpace attribute, the following assertion is added to the Schematron schema in the context of the property, if the tagged value {codeList} has been provided:

*/@codeList={codeList}

To test the existence of the code list value the following assertion is added to the Schematron schema in the context of the property. In {codeListValuePattern} we replace "{codeList}" by {codeList} and "{value}" by "*/@codeListValue":

(not contains('{codeListValuePattern}', '#') and document('{codeListValuePattern}')) or (contains('{codeListValuePattern}', '#') and document(substring-before('{codeListValuePattern}','#'))/id(substring-after('{codeListValuePattern}','#')))

In addition, we can assert that the remote resource has the correct element based on its representation.

For "application/gml+xml;version=3.2" (the default) we expect a gml:Definition:

(not contains('{codeListValuePattern}', '#') and document('{codeListValuePattern}') /gml:Definition) or (contains('{codeListValuePattern}', '#') and document(substring-before('{codeListValuePattern}','#'))/id(substring-after('{codeListValuePattern}','#'))[local-name()='Definiton' and namespace-uri()='http://www.opengis.net/gml/3.2'])

For "application/rdf+xml" we expect a skos:Concept (see the OWS-8 Semantic Mediation ER):

(not contains('{codeListValuePattern}', '#') and document('{codeListValuePattern}')/skos:Concept) or (contains('{codeListValuePattern}', '#') and document(substring-before('{codeListValuePattern}','#'))/id(substring-after('{codeListValuePattern}','#'))[local-name()='Concept' and namespace-uri()=' http://www.w3.org/2004/02/skos/core#'])

6.4.7 Additional assertions on units of measure references

If the schema package has the tagged value uomResourceURI ({uomResourceURI}) and an attribute has the tagged values

Case 1: physicalQuantity ({physicalQuantity}) and recommendedMeasure ({recommendedMeasure}), or

Case 2: noncomparableMeasure ({noncomparableMeasure})

then the following assertion is added to the Schematron schema in the context of the property.

NOTE All assertions assume the default values for the tagged values uomResourceValuePattern and uomResourceRepresentation listed in Table 2. If these differ, the assertions have to be adapted accordingly.

Case 1: @uom='{uomResourceURI}/{physicalQuantity}/{recommendedMeasure}'

Case 2: @uom='{uomResourceURI}/noncomparable/{noncomparableMeasure}

To test the existence of the unit the following assertion is added to the Schematron schema in the context of the property:

Case 1/2: document(@uom)

In addition, we can assert that the remote resource has the correct element based on its representation. I.e., for a GML representation we expect a gml:BaseUnit, gml:Derived or gml:ConventionalUnit:

Case 1/2: $A/gml:BaseUnit|$A/gml:Derived|$A/gml:ConventionalUnit with a let expression to set $A=document(@uom)

6.5 Conclusions from testing the improvements

The planned improvements have been specified, implemented and tested.

In implementation, the decision in ISO 19115 to allow that free text values can be constrained in profiles to values from a code list and break UML created issues in the implementation as ISO-19115-specific code had to be added to the underlying OCL parser. From an implementation perspective it would be important to conform to UML. However, this would require a change in ISO 19115.

7 KML encoding rule improvements

7.1 KML encoding support for different styles per feature type

7.1.1 Overview

The OWS-7 KML encoding rule does not support different styles per feature type. It is required a more refined mechanism to distinguish for example different building symbology based on building functions or conditions. This shortcoming is addressed by the approach described in the following sub-clauses.

7.1.2 Changes to the ShapeChange configuration

The URL of the portrayal rule set in the portrayal registry for the feature type styles of the application schema is passed to ShapeChange as a parameter in the configuration file.

```
<targetParameter name="portrayalRuleSetUri" value="<URI of the rule
set>"/>
```

Example:

```
<targetParameter name="portrayalRuleSetUri" value="http://ows8-
cci.carmenta.com/prs/processingservice?request=GetRules&RuleSet=10001&e
ncoding=unresolved-SE"/>
```

7.1.3 Changes to the encoding rule implemented by ShapeChange

In OWS-7, a tagged value was the only way to determine the reference to a kml:Style or kml:StyleMap element. To support instance-specific styling, ShapeChange now supports access to a portrayal rule set in the portrayal registry using the URL in the portrayalRuleSetUri configuration parameter to retrieve the portrayal rules for the relevant feature types and convert the rules to XSLT elements that determine the kml:Style or kml:StyleMap URL for each instance of the feature type. The kml:Style and kml:StyleMap URLs are provided by the portrayal registry. For additional details see the OWS-8 Portrayal Registry Engineering Report (11-062).

As a result, the same portrayal rules are in principle used by WMS/SLD implementations as well as the KML implementation.

The words "in principle" are used here as changes to the standard Symbology Encoding 1.1 schema are required for this. Symbology Encoding currently requires that for each portrayal rule (se:Rule) the symbolisers (se:Symbolizer) are embedded inline. Since the portrayal registry should also centrally host the symboliser information for KML placemarks (kml:Style), the Symbology Encoding schema was amended to allow the decoupling of se:Rule and se:Symbolizer elements. In the schema amended by OWS-8, instead of the 1..n se:Symbolizer elements also a se:OnlineResource element is allowed, referencing a sequence of se:Symbolizer elements or a KML document with a kml:Style element based on the symbolisers.

Example:

A portrayal rule for tds:BuildingGeopoint features in the rule set used in OWS-8 is

```
<se:Rule>
    <ogc:Filter xmlns:tds="http://metadata.dod.mil/mdr/ns/GSIP/3.0/tds/3
    .0" xmlns:ogc="http://www.opengis.net/ogc">
        <ogc:And>
            <ogc:Or>
                <ogc:PropertyIsEqualTo>
                    <ogc:PropertyName>tds:featureFunction-
                    1</ogc:PropertyName>
                    <ogc:Literal>inPatientCare</ogc:Literal>
                </ogc:PropertyIsEqualTo>
                <ogc:PropertyIsEqualTo>
                    <ogc:PropertyName>tds:featureFunction-
                    1</ogc:PropertyName>
                    <ogc:Literal>urgentMedicalCare</ogc:Literal>
                </ogc:PropertyIsEqualTo>
            </ogc:Or>
            <ogc:PropertyIsLessThan>
                <ogc:PropertyName>tds:heightAboveSurfaceLevel</ogc:Propert
                yName>
                <ogc:Literal>46</ogc:Literal>
            </ogc:PropertyIsLessThan>
        </ogc:And>
    </ogc:Filter>
    <se:PointSymbolizer xmlns:se="http://www.opengis.net/se">
        <se:Graphic>
            <se:ExternalGraphic>
                <se:OnlineResource xmlns:xlink="http://www.w3.org/1999/xli
                nk" xlink:type="simple" xlink:href="http://ows8-
                cci.carmenta.com/symbols/hospital.svg"/>
                <se:Format>image/svg+xml</se:Format>
            </se:ExternalGraphic>
        </se:Graphic>
    </se:PointSymbolizer>
</se:Rule>
```

This is the representation using the unamended Symbology Encoding schema. The amended representation is as follows:

```
<se:Rule>
    <ogc:Filter xmlns:tds="http://metadata.dod.mil/mdr/ns/GSIP/3.0/tds/3
    .0" xmlns:ogc="http://www.opengis.net/ogc">
        <ogc:And>
            <ogc:Or>
                <ogc:PropertyIsEqualTo>
                    <ogc:PropertyName>tds:featureFunction-
                    1</ogc:PropertyName>
                    <ogc:Literal>inPatientCare</ogc:Literal>
                </ogc:PropertyIsEqualTo>
                <ogc:PropertyIsEqualTo>
                    <ogc:PropertyName>tds:featureFunction-
                    1</ogc:PropertyName>
                    <ogc:Literal>urgentMedicalCare</ogc:Literal>
                </ogc:PropertyIsEqualTo>
            </ogc:Or>
            <ogc:PropertyIsLessThan>
                <ogc:PropertyName>tds:heightAboveSurfaceLevel</ogc:Propert
                yName>
                <ogc:Literal>46</ogc:Literal>
            </ogc:PropertyIsLessThan>
        </ogc:And>
    </ogc:Filter>
    <se:OnlineResource xlink:type="simple" xlink:href="http://ows8-
    cci.carmenta.com/ProcessingService/ProcessingService.axd?request=Get
    Symbols&Symbol=219"/>
</se:Rule>
```

where the referenced symbol is

```
<SymbolizerList xmlns:sld="http://www.opengis.net/sld" xmlns="http://ww
w.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"xmlns:se="http
://www.opengis.net/se" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" symbolId="219"xsi:schemaLocation="http://schemas.opengis.net/
sld/1.1/StyledLayerDescriptor.xsd">
    <se:PointSymbolizer xmlns:se="http://www.opengis.net/se">
        <se:Graphic>
            <se:ExternalGraphic>
                <se:OnlineResource xmlns:xlink="http://www.w3.org/1999/xli
                nk" xlink:type="simple" xlink:href="http://ows8-
                cci.carmenta.com/symbols/hospital.svg"/>
                <se:Format>image/svg+xml</se:Format>
            </se:ExternalGraphic>
        </se:Graphic>
    </se:PointSymbolizer>
</SymbolizerList>
```

sld:SymbolizerList is not a standard SLD element, but another schema amendment of the OWS-8 portrayal registry.

To request the KML representation from the portrayal registry processing service, a parameter "encoding=application/vnd.google-earth.kml+xml" has to be appended to URL. I.e.,

http://ows8-cci.carmenta.com/ProcessingService/ProcessingService.axd?request=GetSymbols&Symbol=219&encoding=application/vnd.google-earth.kml+xml

returns

```
<kml
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.opengis.net/kml/2.2">
<Document>
```

```
    <Style id="219">
        <IconStyle>
            <color>FFFFFFFF</color>
            <colorMode>normal</colorMode>
            <scale>1</scale>
            <heading>0</heading>
            <Icon>
                <href>http://ows8-
                cci.carmenta.com/symbols/hospital.svg</href>
            </Icon>
        </IconStyle>
    </Style>
</Document>
</kml>
```

When the XSLT template for a feature type is created, all portrayal rules for that feature type in the rule set are collected and processed in reverse order (the order of rules in the rule set follows the painters model, i.e. later rules are applied/drawn on top of the earlier rules). After the kml:visibility element, XSLT elements are inserted to select the appropriate kml:Style reference. For this, the Filter is converted into an Xpath expression. The current implementation supports filter expressions that use the logical operators and the binary comparison operators (comparison of feature properties with literal values).

Example:

For the tds:BuildingGeopoint features, the resulting XSLT elements are as follows (the sample rule shown in the previous example is highlighted in blue):

```
<choose xmlns="http://www.w3.org/1999/XSL/Transform">
    <when test="tds:heightAboveSurfaceLevel &gt;= 46">
        <styleUrl xmlns="http://www.opengis.net/kml/2.2">http://ows8-
        cci.carmenta.com/ProcessingService/ProcessingService.axd?request=
        GetSymbols%26Symbol=211%26Encoding=application/vnd.google-
        earth.kml+xml#211</styleUrl>
    </when>
    <when test=" ( (tds:featureFunction-1 = 'education' or
    tds:featureFunction-1 = 'primaryEducation' or tds:featureFunction-1
    = 'secondaryEducation' or tds:featureFunction-1 = 'higherEducation'
    or tds:featureFunction-1 = 'vocationalEducation' or
    tds:featureFunction-1 = 'institution')  and
    tds:heightAboveSurfaceLevel &lt; 46) ">
        <styleUrl xmlns="http://www.opengis.net/kml/2.2">http://ows8-
        cci.carmenta.com/ProcessingService/ProcessingService.axd?request=
        GetSymbols%26Symbol=220%26Encoding=application/vnd.google-
        earth.kml+xml#220</styleUrl>
    </when>
    <when test=" ( (tds:featureFunction-1 = 'inPatientCare' or
    tds:featureFunction-1 = 'urgentMedicalCare')  and
    tds:heightAboveSurfaceLevel &lt; 46) ">
        <styleUrl xmlns="http://www.opengis.net/kml/2.2">http://ows8-
        cci.carmenta.com/ProcessingService/ProcessingService.axd?request=
        GetSymbols%26Symbol=219%26Encoding=application/vnd.google-
        earth.kml+xml#219</styleUrl>
    </when>
    <when test="tds:heightAboveSurfaceLevel &lt; 46">
```

```
      <styleUrl xmlns="http://www.opengis.net/kml/2.2">http://ows8-
      cci.carmenta.com/ProcessingService/ProcessingService.axd?request=
      GetSymbols%26Symbol=210%26Encoding=application/vnd.google-
      earth.kml+xml#210</styleUrl>
   </when>
</choose>
```

This results in the appropriate kml:styleUrl being added to the kml:placemark created for the feature.

The portrayal rule set used in OWS-8 did not contain any scale range information. As a result, the encoding rule does not consider scale.

7.1.4 Improved KML encoding rule

The resulting encoding rule, which is a slightly amended version of the OWS-7 KML encoding rule, is specified in Annex B.

The mapping of data conforming to an ISO 19109 conformant UML Application Schema to a KML representation is based on a set of encoding rules. These encoding rules are compliant with the rules for KML and ISO 19118.

Compared to the GML encoding rule specified in GML 3.2 Annex E, the KML encoding rule is different, which reflects the different characteristics of GML and KML. In particular, no XML Schema description is derived for the KML encoding.

The rules listed in Annex B aim at an automatic mapping from an ISO 19109 and ISO/TS 19103 conformant UML application schema to KML. As a result of this automation, the resulting KML does not make full use of the capabilities of KML.

The rules for the instance conversion, as documented in Annex B, are worded so that GML data is assumed as input. This reflects the implementation where KML is an additional output format of a Web Feature Service, which always has to support GML, too.

The schema encoding rules are based on the general idea that all features conforming to a feature type in the application schema are represented as KML placemarks, additional information is represented in kml:ExtendedData elements and the style may be determined by a portrayal rule set in an amended Symbology Encoding schema.

The encoding rule has been designed with the goal to maximize the use of standard capabilities of KML 2.2 and of existing clients with a focus on Google Earth as the standard client for using KML data. Extensions not supported by Google Earth or other clients have been avoided, whenever possible.

7.2 Evaluate the use of portrayal registries within KML

7.2.1 Overview and summary

The approach documented in 7.1 has been tested with

- the OWS-8 portrayal registry provided by Carmenta
- the OWS-8 LTDS Web Feature Service provided by interactive instruments
- the KML cache of the LTDS Web Feature Service data (see 7.3)

The XSLT stylesheet is accessed by the WFS whenever the GetFeature request uses the outputFormat parameter with the value application/vnd.google-earth.kml+xml. The generation of the XSLT stylesheet can be automated to occur periodically – or, if a notification mechanism of the portrayal registry exists, whenever relevant changes occur in the portrayal registry.

The following figures show example screenshots of the OWS-8 test data in Google Earth and also of other data offered by Google Earth "out-of-the-box". The figures cover a range of LTDS feature types. For buildings and roads the figures also illustrates different styles for placemarks even though they were derived from the same LTDS feature type.

A number of issues were identified in the tests. Some issues could be improved through refinements in the implementations or the encoding rule for the particular data set, while some are of a conceptual nature and are general limitations of the approach. These issues are discussed in 7.2.2, 7.2.3 and 7.2.4.

Note that the name of the features is displayed in the balloons as "no Information". The reason is that this is the value of the name property of the source features in the data set.

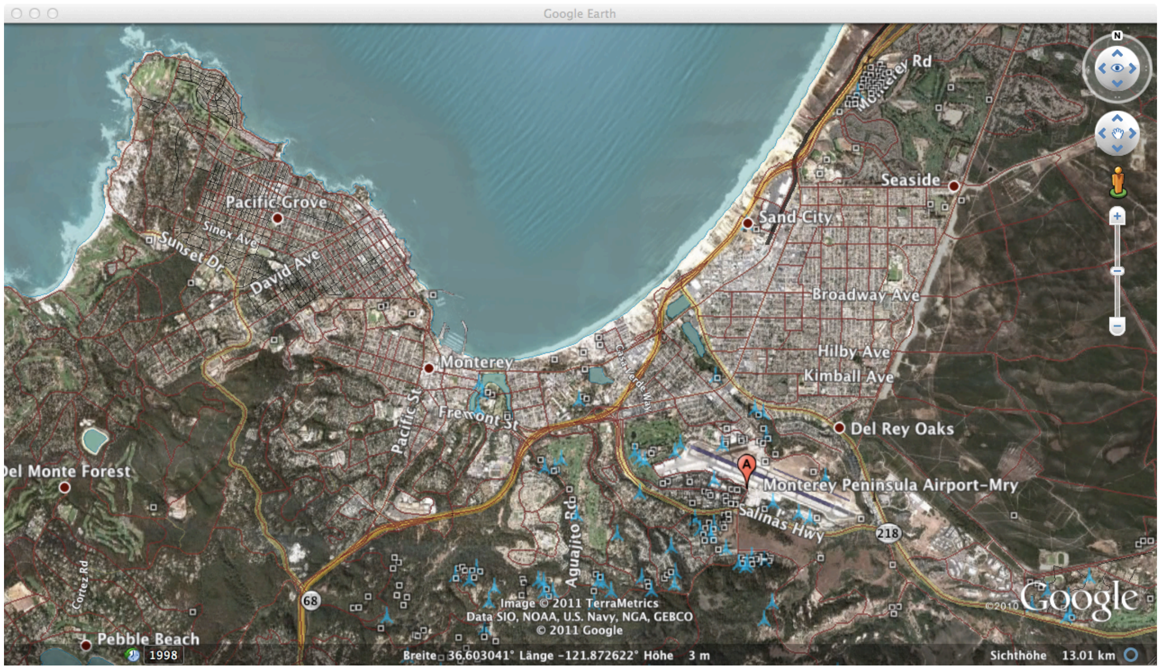
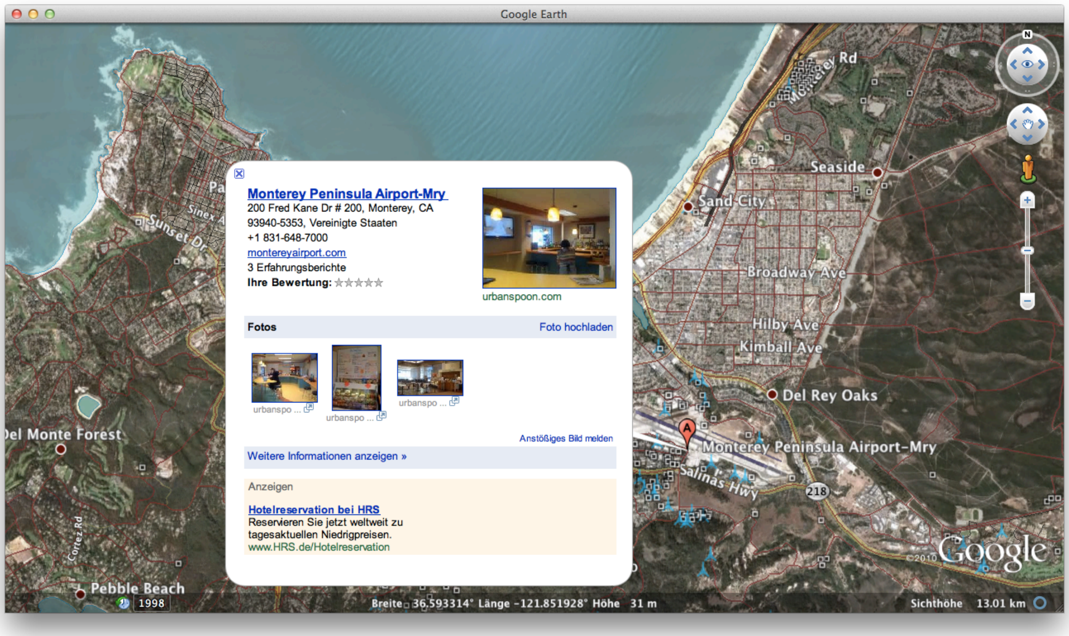


Figure 7 – Overview Monterey area



Monterey Peninsula Airport-Mry
200 Fred Kane Dr # 200, Monterey, CA
93940-5353, Vereinigte Staaten
+1 831-648-7000
montereyairport.com
3 Erfahrungsberichte
Ihre Bewertung: ☆☆☆☆☆
urbanspoon.com

Fotos Foto hochladen

urbanspo ... urbanspo ... urbanspo ...

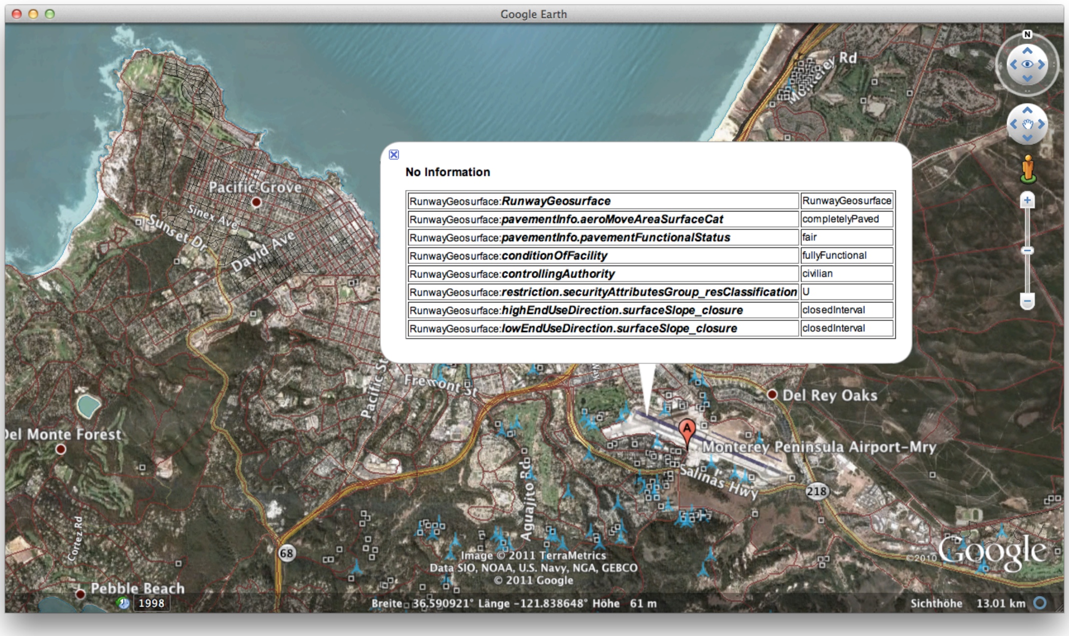
Anstößiges Bild melden

Weitere Informationen anzeigen »

Anzeigen

Hotelreservation bei HRS
Reservieren Sie jetzt weltweit zu
tagesaktuellen Niedrigpreisen.
www.HRS.de/Hotelreservation

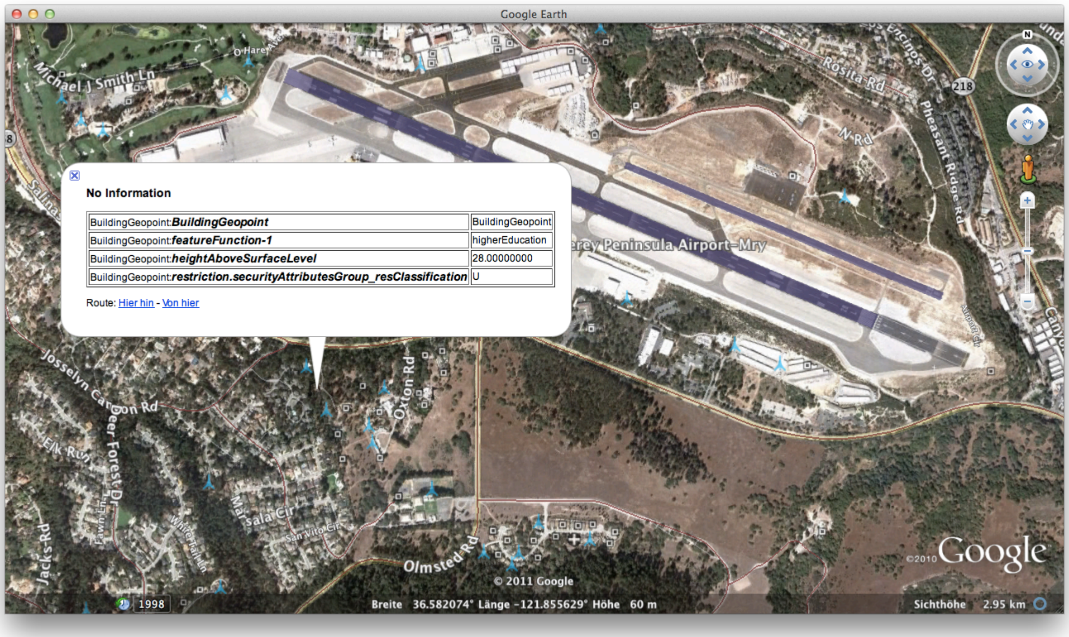
Figure 8 – Information about Monterey Airport offered by Google Earth



No Information

RunwayGeosurface:***RunwayGeosurface***	RunwayGeosurface
RunwayGeosurface:***pavementInfo.aeroMoveAreaSurfaceCat***	completelyPaved
RunwayGeosurface:***pavementInfo.pavementFunctionalStatus***	fair
RunwayGeosurface:***conditionOfFacility***	fullyFunctional
RunwayGeosurface:***controllingAuthority***	civilian
RunwayGeosurface:***restriction.securityAttributesGroup_resClassification***	U
RunwayGeosurface:***highEndUseDirection.surfaceSlope_closure***	closedInterval
RunwayGeosurface:***lowEndUseDirection.surfaceSlope_closure***	closedInterval

Figure 9 – A runway, the information shown is LTDS data

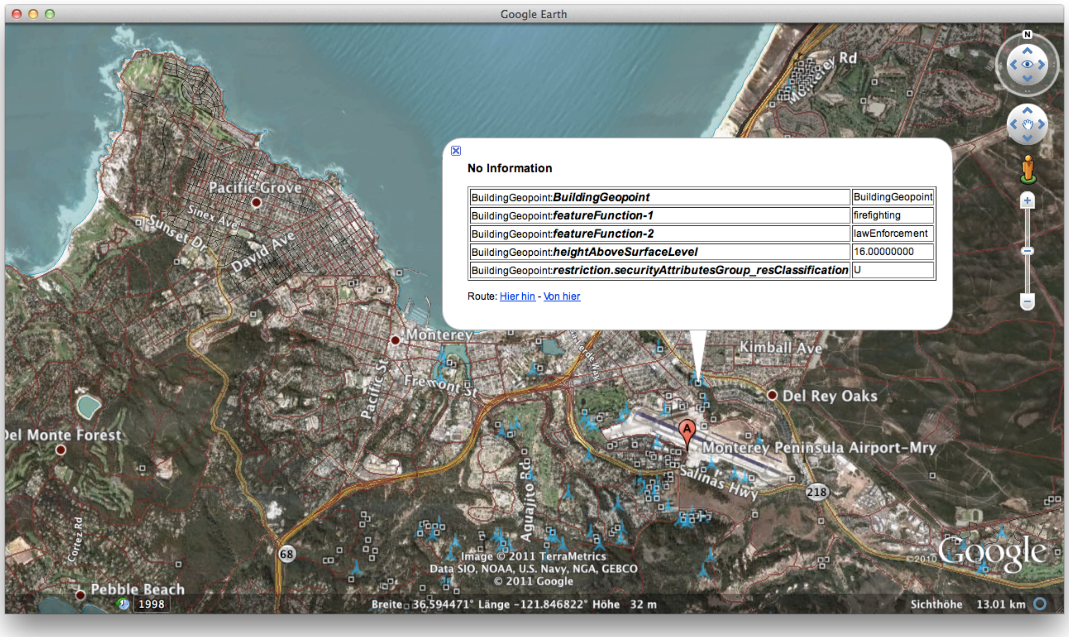


No Information

BuildingGeopoint:***BuildingGeopoint***	BuildingGeopoint
BuildingGeopoint:***featureFunction-1***	higherEducation
BuildingGeopoint:***heightAboveSurfaceLevel***	28.00000000
BuildingGeopoint:***restriction.securityAttributesGroup_resClassification***	U

Route: Hier hin - Von hier

Figure 10 – An education building, height < 46, the information shown is LTDS data

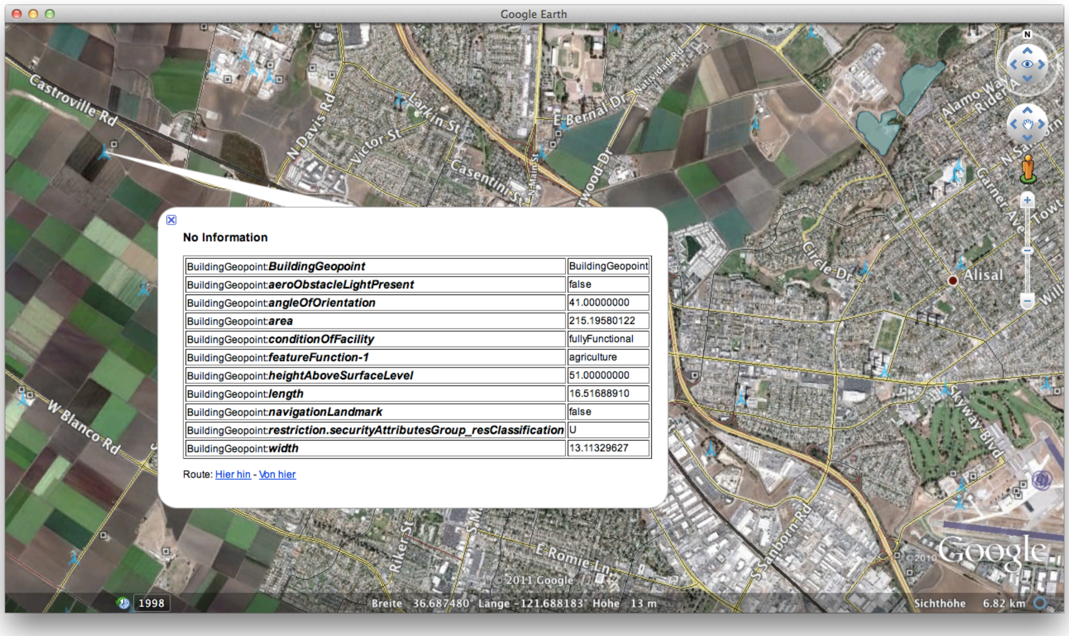


No Information

BuildingGeopoint:***BuildingGeopoint***	BuildingGeopoint
BuildingGeopoint:***featureFunction-1***	firefighting
BuildingGeopoint:***featureFunction-2***	lawEnforcement
BuildingGeopoint:***heightAboveSurfaceLevel***	16.00000000
BuildingGeopoint:***restriction.securityAttributesGroup_resClassification***	U

Route: Hier hin - Von hier

Figure 11 – Another building with height < 46, the information shown is LTDS data

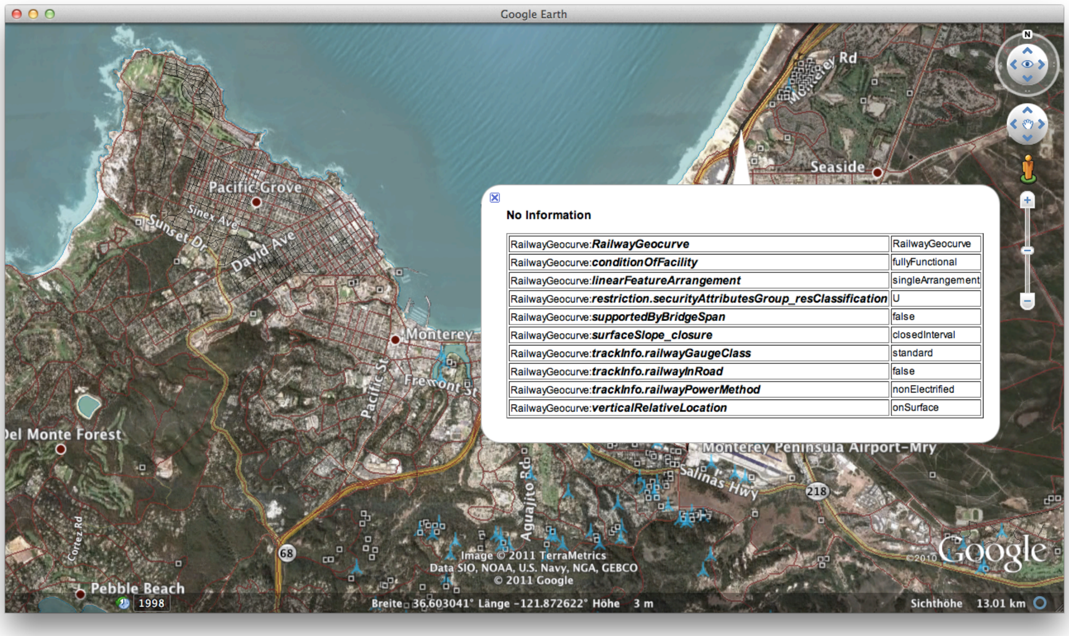


No Information

BuildingGeopoint:***BuildingGeopoint***	BuildingGeopoint
BuildingGeopoint:***aeroObstacleLightPresent***	false
BuildingGeopoint:***angleOfOrientation***	41.00000000
BuildingGeopoint:***area***	215.19580122
BuildingGeopoint:***conditionOfFacility***	fullyFunctional
BuildingGeopoint:***featureFunction-1***	agriculture
BuildingGeopoint:***heightAboveSurfaceLevel***	51.00000000
BuildingGeopoint:***length***	16.51688910
BuildingGeopoint:***navigationLandmark***	false
BuildingGeopoint:***restriction.securityAttributesGroup_resClassification***	U
BuildingGeopoint:***width***	13.11329627

Route: Hier hin - Von hier

Figure 12 – Another building, height > 46, the information shown is LTDS data



No Information

RailwayGeocurve:***RailwayGeocurve***	RailwayGeocurve
RailwayGeocurve:***conditionOfFacility***	fullyFunctional
RailwayGeocurve:***linearFeatureArrangement***	singleArrangement
RailwayGeocurve:***restriction.securityAttributesGroup_resClassification***	U
RailwayGeocurve:***supportedByBridgeSpan***	false
RailwayGeocurve:***surfaceSlope_closure***	closedInterval
RailwayGeocurve:***trackInfo.railwayGaugeClass***	standard
RailwayGeocurve:***trackInfo.railwayInRoad***	false
RailwayGeocurve:***trackInfo.railwayPowerMethod***	nonElectrified
RailwayGeocurve:***verticalRelativeLocation***	onSurface

Figure 13 – A railway, the information shown is LTDS data

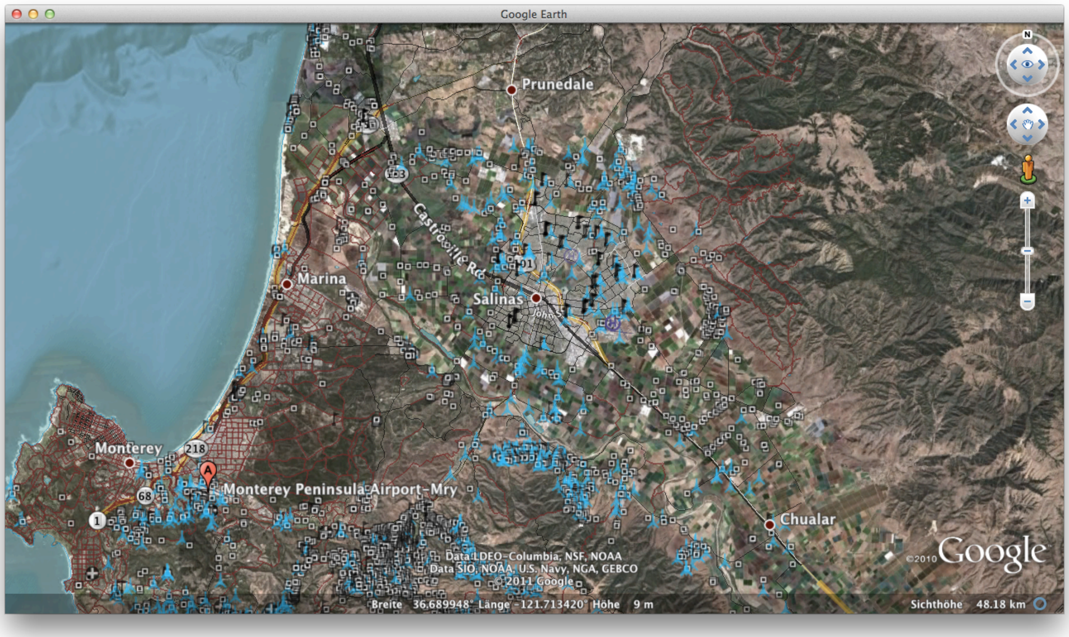
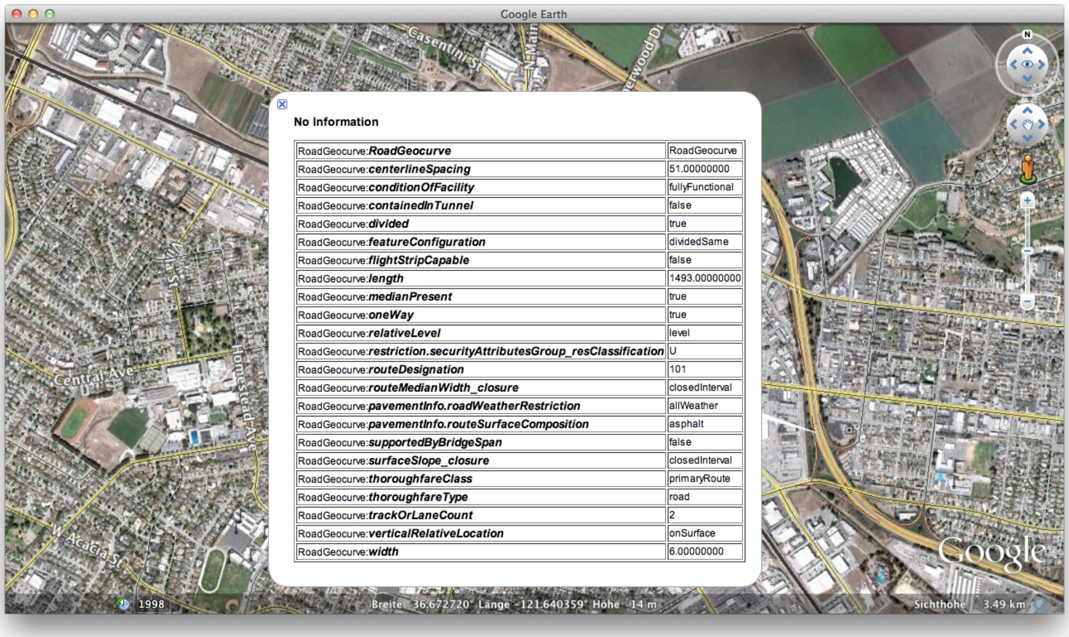

Google Earth
Prunedale
Marina
Castroville Rd
Salinas
Monterey
Monterey Peninsula Airport-Mry
Chualar
Data LDEO-Columbia, NSF, NOAA
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
© 2011 Google
Google
Breite 36.689948° Länge -121.713420° Höhe 9 m
Sichthöhe 48.18 km

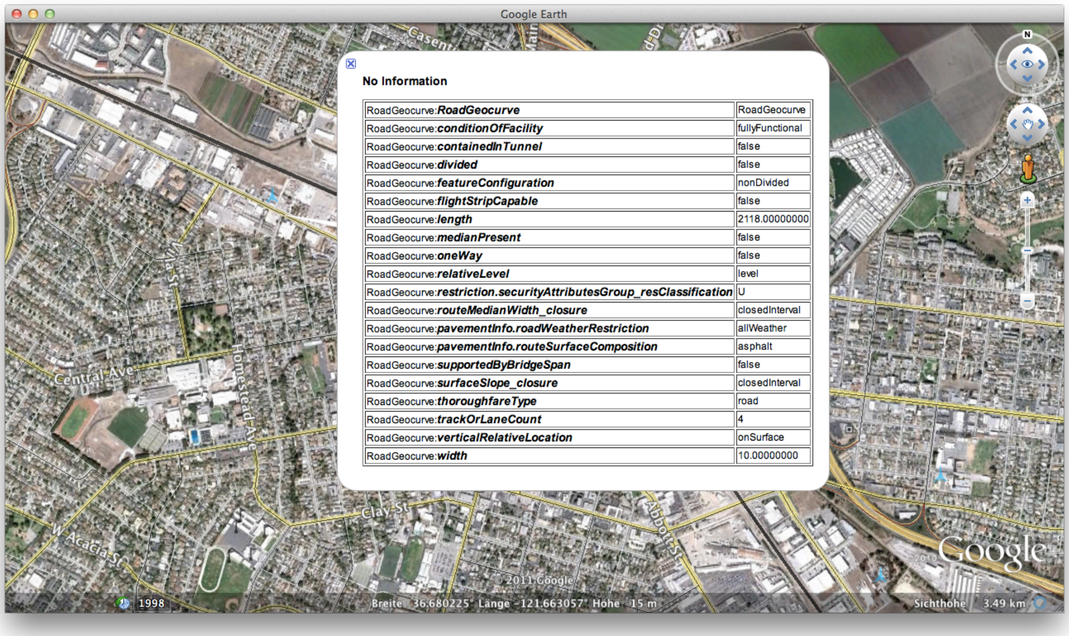
Figure 14 – Monterey bay area with Salinas (and a number of educational buildings)



No Information

RoadGeocurve:***RoadGeocurve***	RoadGeocurve
RoadGeocurve:***centerlineSpacing***	51.00000000
RoadGeocurve:***conditionOfFacility***	fullyFunctional
RoadGeocurve:***containedInTunnel***	false
RoadGeocurve:***divided***	true
RoadGeocurve:***featureConfiguration***	dividedSame
RoadGeocurve:***flightStripCapable***	false
RoadGeocurve:***length***	1493.00000000
RoadGeocurve:***medianPresent***	true
RoadGeocurve:***oneWay***	true
RoadGeocurve:***relativeLevel***	level
RoadGeocurve:***restriction.securityAttributesGroup_resClassification***	U
RoadGeocurve:***routeDesignation***	101
RoadGeocurve:***routeMedianWidth_closure***	closedInterval
RoadGeocurve:***pavementInfo.roadWeatherRestriction***	allWeather
RoadGeocurve:***pavementInfo.routeSurfaceComposition***	asphalt
RoadGeocurve:***supportedByBridgeSpan***	false
RoadGeocurve:***surfaceSlope_closure***	closedInterval
RoadGeocurve:***thoroughfareClass***	primaryRoute
RoadGeocurve:***thoroughfareType***	road
RoadGeocurve:***trackOrLaneCount***	2
RoadGeocurve:***verticalRelativeLocation***	onSurface
RoadGeocurve:***width***	6.00000000

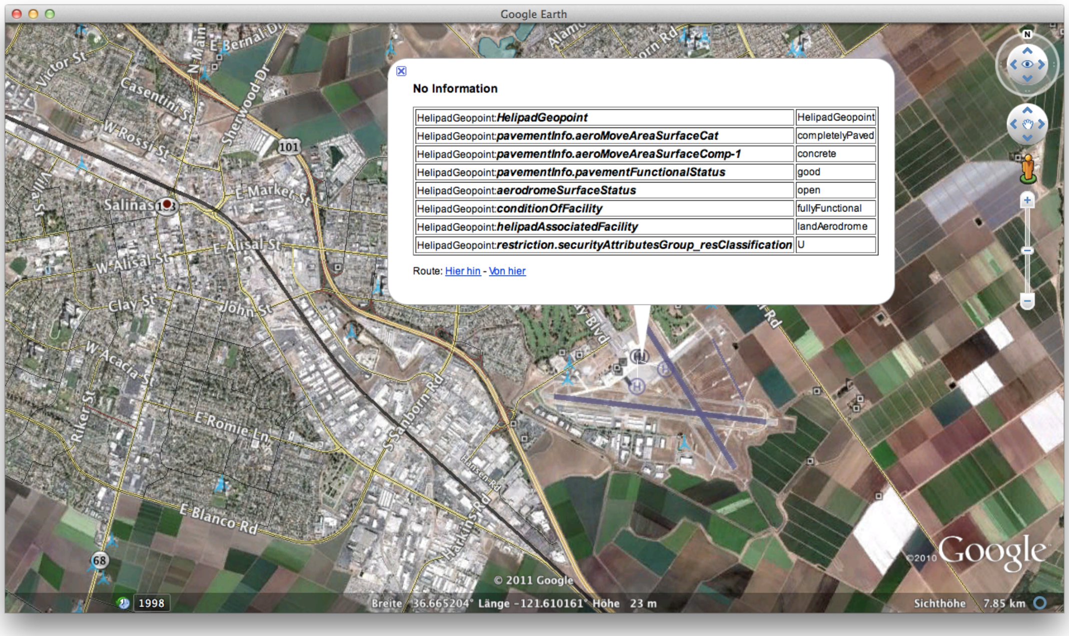
Figure 15 – A road with median, the information shown is LTDS data



No Information

RoadGeocurve:***RoadGeocurve***	RoadGeocurve
RoadGeocurve:***conditionOfFacility***	fullyFunctional
RoadGeocurve:***containedInTunnel***	false
RoadGeocurve:***divided***	false
RoadGeocurve:***featureConfiguration***	nonDivided
RoadGeocurve:***flightStripCapable***	false
RoadGeocurve:***length***	2118.00000000
RoadGeocurve:***medianPresent***	false
RoadGeocurve:***oneWay***	false
RoadGeocurve:***relativeLevel***	level
RoadGeocurve:***restriction.securityAttributesGroup_resClassification***	U
RoadGeocurve:***routeMedianWidth_closure***	closedInterval
RoadGeocurve:***pavementInfo.roadWeatherRestriction***	allWeather
RoadGeocurve:***pavementInfo.routeSurfaceComposition***	asphalt
RoadGeocurve:***supportedByBridgeSpan***	false
RoadGeocurve:***surfaceSlope_closure***	closedInterval
RoadGeocurve:***thoroughfareType***	road
RoadGeocurve:***trackOrLaneCount***	4
RoadGeocurve:***verticalRelativeLocation***	onSurface
RoadGeocurve:***width***	10.00000000

Figure 16 – A road without median, the information shown is LTDS data

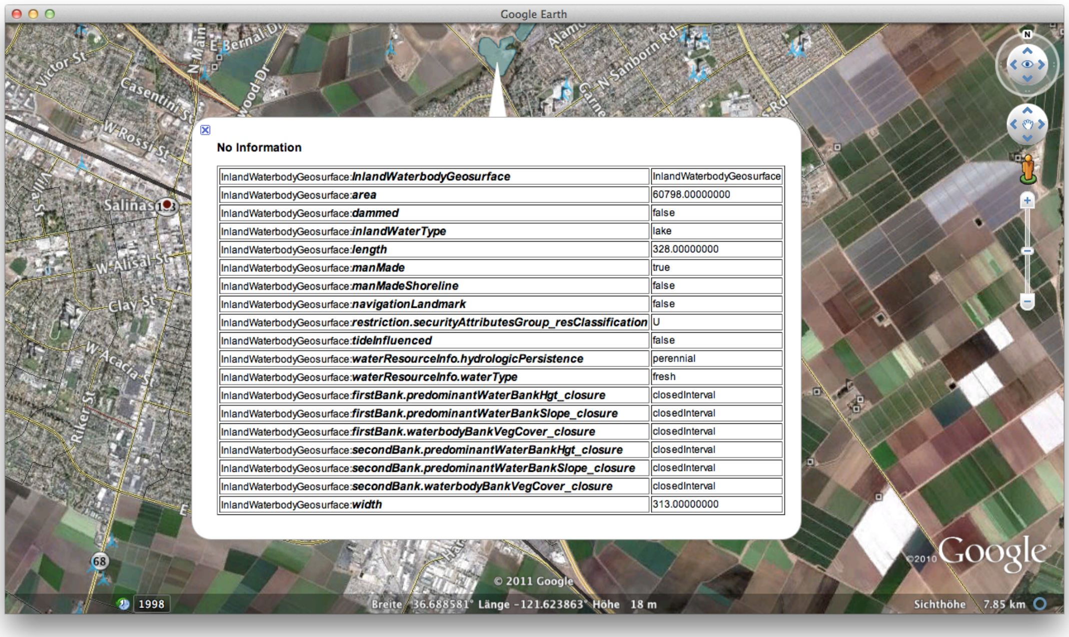


No Information

HelipadGeopoint:***HelipadGeopoint***	HelipadGeopoint
HelipadGeopoint:***pavementInfo.aeroMoveAreaSurfaceCat***	completelyPaved
HelipadGeopoint:***pavementInfo.aeroMoveAreaSurfaceComp-1***	concrete
HelipadGeopoint:***pavementInfo.pavementFunctionalStatus***	good
HelipadGeopoint:***aerodromeSurfaceStatus***	open
HelipadGeopoint:***conditionOfFacility***	fullyFunctional
HelipadGeopoint:***helipadAssociatedFacility***	landAerodrome
HelipadGeopoint:***restriction.securityAttributesGroup_resClassification***	U

Route: Hier hin - Von hier

Figure 17 – A heliport, the information shown is LTDS data



No Information

InlandWaterbodyGeosurface:***InlandWaterbodyGeosurface***	InlandWaterbodyGeosurface
InlandWaterbodyGeosurface:***area***	60798.00000000
InlandWaterbodyGeosurface:***dammed***	false
InlandWaterbodyGeosurface:***inlandWaterType***	lake
InlandWaterbodyGeosurface:***length***	328.00000000
InlandWaterbodyGeosurface:***manMade***	true
InlandWaterbodyGeosurface:***manMadeShoreline***	false
InlandWaterbodyGeosurface:***navigationLandmark***	false
InlandWaterbodyGeosurface:***restriction.securityAttributesGroup_resClassification***	U
InlandWaterbodyGeosurface:***tideInfluenced***	false
InlandWaterbodyGeosurface:***waterResourceInfo.hydrologicPersistence***	perennial
InlandWaterbodyGeosurface:***waterResourceInfo.waterType***	fresh
InlandWaterbodyGeosurface:***firstBank.predominantWaterBankHgt_closure***	closedInterval
InlandWaterbodyGeosurface:***firstBank.predominantWaterBankSlope_closure***	closedInterval
InlandWaterbodyGeosurface:***firstBank.waterbodyBankVegCover_closure***	closedInterval
InlandWaterbodyGeosurface:***secondBank.predominantWaterBankHgt_closure***	closedInterval
InlandWaterbodyGeosurface:***secondBank.predominantWaterBankSlope_closure***	closedInterval
InlandWaterbodyGeosurface:***secondBank.waterbodyBankVegCover_closure***	closedInterval
InlandWaterbodyGeosurface:***width***	313.00000000

Figure 18 – A surface waterbody, the information shown is LTDS data

7.2.2 Summary

The tests have highlighted a number of issues that are discussed in the following sub-clauses, separated into conceptual issues and implementation issues and limitations.

One of the goals of the experiments in OWS-8 were to "evaluate and demonstrate if the same style resources could be used for OGC Portrayal Services and KML data, i.e. KML and SLD/SE would simply be different representations of the same resource." The conclusion from the experiments is that this is only possible to a certain extent. In practice this will likely only work for very simple portrayal rule sets.

7.2.3 Conceptual issues

7.2.3.1 One feature, multiple applicable rules

For the same feature instance, several rules in a portrayal rule set may apply. I.e., a standard Symbology Encoding processor will render the same feature according to multiple rules on a map, if several rules apply for the specific feature instance.

In KML a placemark can reference only a single style. The current implementation uses the topmost portrayal rule that matches the feature instance (which is the last portrayal rule in the portrayal rule set following the painters model). All other rules that would fire are not reflected in the KML.

This does not seem to be a significant issue in the simple rule set used in OWS-8.

Example:

```
<se:FeatureTypeStyle>
   <se:FeatureTypeName>BuildingGeopoint</se:FeatureTypeName>
   <se:Rule>
      <ogc:Filter xmlns:tds="http://metadata.dod.mil/mdr/ns/GSIP/3.0/td
      s/3.0" xmlns:ogc="http://www.opengis.net/ogc">
         <ogc:PropertyIsLessThan>
            <ogc:PropertyName>tds:heightAboveSurfaceLevel</ogc:Propert
            yName>
            <ogc:Literal>46</ogc:Literal>
         </ogc:PropertyIsLessThan>
      </ogc:Filter>
      <se:OnlineResource xlink:type="simple" xlink:href="http://ows8-
      cci.carmenta.com/ProcessingService/ProcessingService.axd?request=
      GetSymbols&Symbol=210"/>
   </se:Rule>
   <se:Rule>
      <ogc:Filter xmlns:tds="http://metadata.dod.mil/mdr/ns/GSIP/3.0/td
      s/3.0" xmlns:ogc="http://www.opengis.net/ogc">
         <ogc:And>
            <ogc:Or>
               <ogc:PropertyIsEqualTo>
                  <ogc:PropertyName>tds:featureFunction-
                  1</ogc:PropertyName>
                  <ogc:Literal>inPatientCare</ogc:Literal>
               </ogc:PropertyIsEqualTo>
               <ogc:PropertyIsEqualTo>
                  <ogc:PropertyName>tds:featureFunction-
                  1</ogc:PropertyName>
                  <ogc:Literal>urgentMedicalCare</ogc:Literal>
               </ogc:PropertyIsEqualTo>
            </ogc:Or>
            <ogc:PropertyIsLessThan>
               <ogc:PropertyName>tds:heightAboveSurfaceLevel</ogc:Prop
               ertyName>
               <ogc:Literal>46</ogc:Literal>
            </ogc:PropertyIsLessThan>
         </ogc:And>
      </ogc:Filter>
      <se:OnlineResource xlink:type="simple" xlink:href="http://ows8-
      cci.carmenta.com/ProcessingService/ProcessingService.axd?request=
      GetSymbols&Symbol=219"/>
   </se:Rule>
   ...
</se:FeatureTypeStyle>
```

For a BuildingGeopoint feature with tds:heightAboveSurfaceLevel with a value of 40 and a tds:featureFunction-1 of inPatientCare, both the first and second rule would "fire" and both symbolisers will be drawn by a WMS/SLD. In KML, the feature would only be rendered using symbol (219) of the second rule, the topmost of the "firing" rules.

7.2.3.2 Painters model

This issue is related to the previous issue. In Symbology Encoding, a feature collection is processed multiple times and feature instances are rendered according to the painters model. This determines the order in which symbols are visible on the map. KML on the other hand has no such concept and the order in which features are rendered by a KML client per se cannot be controlled.

An assumption might be that the features will be styled by a KML client in the order in which they appear in a KML document, but this is not required and not something that can be relied upon. However, let's assume for a moment that this is how all KML clients would work. In this case two aspects would be noteworthy:

- Ordering the placemarks in a KML document in accordance with the order implied by the portrayal rule set requires several passes through the feature collection, one for each level in the painters model. This has performance implications, which can be minimised, for example, by using a cache like the one described in 7.3.
- Once the processing of the rule set becomes the most complex part of the transformation of the feature collection into KML, then it is probably more appropriate to perform this transformation in a WMS/SLD as in this case the focus is less on the data aspect and more on the portrayal aspect.
  In OWS-7 the focus of the KML output was on representing as good as possible the information associated with a feature in an appropriate way in a KML client. Hence, the KML was transformed by the WFS, if the outputFormat parameter was set to the KML MIME type. As long as processing portrayal rules is a minor part of the transformation this still seems appropriate as the focus is on the data carried with the KML placemarks. However, if the transformation would have to honour the painters model, processing requires a full Symbology Encoding processing engine and as a result this transformation should be carried out by a WMS/SLD - which might be a component WMS/SLD (FPS) or an integrated WMS/SLD.

7.2.3.3 KML styles simpler than SE symboliser sets

Complex symboliser sets in a Symbology Encoding rule cannot be transformed entirely into KML styles. For example, multiple line symbolisers with different width and colour that are often used for higher category roads cannot be properly transformed into KML styles as a KML style can include only a single line style.

Example:

The following sequence of symbolisers in a portrayal rule for tds:RoadGeocurve features cannot be transformed properly into a kml:Style as a kml:Style may contain only a single kml:LineStyle):

```
<se:LineSymbolizer xmlns:se="http://www.opengis.net/se">
   <se:Name>Black_1.45mmSolidLine</se:Name>
   <se:Stroke>
      <se:SvgParameter name="stroke">#000000</se:SvgParameter>
      <se:SvgParameter name="stroke-width">7</se:SvgParameter>
   </se:Stroke>
   </se:LineSymbolizer>
<se:LineSymbolizer xmlns:se="http://www.opengis.net/se">
   <se:Name>Dk-Brown1815_1.15mmSolidLine</se:Name>
   <se:Stroke>
      <se:SvgParameter name="stroke">#782327</se:SvgParameter>
      <se:SvgParameter name="stroke-width">5</se:SvgParameter>
   </se:Stroke>
</se:LineSymbolizer>
<se:LineSymbolizer xmlns:se="http://www.opengis.net/se">
   <se:Name>Black0_0.55mmSolidLine</se:Name>
   <se:Stroke>
      <se:SvgParameter name="stroke">#000000</se:SvgParameter>
      <se:SvgParameter name="stroke-width">3</se:SvgParameter>
   </se:Stroke>
</se:LineSymbolizer>
<se:LineSymbolizer xmlns:se="http://www.opengis.net/se">
   <se:Name>PaperWhite_0.25mmSolidLine</se:Name>
   <se:Stroke>
      <se:SvgParameter name="stroke">#FFFFFF</se:SvgParameter>
      <se:SvgParameter name="stroke-width">1</se:SvgParameter>
   </se:Stroke>
</se:LineSymbolizer>
```

The transformed kml:Style provided by the portrayal registry is (using only the last, i.e., topmost se:LineSymbolizer):

```
<kml xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns="http://www
.opengis.net/kml/2.2">
<Document>
   <Style id="214">
      <LineStyle>
         <color>FF000000</color>
         <colorMode>normal</colorMode>
         <width>1</width>
      </LineStyle>
   </Style>
</Document>
</kml>
```

As a result, the road selected in Figure 15 is in KML in a very simple style, while a WMS/SLD would draw a more complex symbol.

7.2.3.4 KML supports interaction with the map, SE does not

One of the characteristics of KML is that it supports the interaction of the user with the map and its content. For example, if a mouse moves over a feature (placemark), the feature can change into a different style. As Symbology Encoding was designed with static bitmap images in mind, interactions with the map are not supported in the Symbology Encoding design. This is not a problem per se, but a KML representation of a set of Symbology Encoding symbolisers does not fully exploit the capabilities of KML and may not meet the expectation of users as the behaviour of such KML data is different from other typical KML data. I.e., if we want to use Symbology Encoding as the basis for portrayal rules for KML output, Symbology Encoding should be enriched to support feature highlighting.

This could be achieved, for example, with a new rule element that is used for highlighting, e.g. sex:RuleHighlighted. Such element could be substitutable for se:Rule and have the same content model.

7.2.4 Implementation issues and limitations

7.2.4.1 Overview

In the process of the interoperability testing using Google Earth as a client accessing KML data from the LTDS WFS via the KML cache and accessing the symbols from the portrayal registry processing service several issues were identified and resolved. They are listed here as others might learn from these experiences, too.

7.2.4.2 Support for SVG icons

The standard icon symbols in the portrayal registry are in SVG and they need to be converted, for example, to PNG for using them in a KML context as at least Google Earth does not support SVG icons at the moment. An issue for this has already been registered, see http://code.google.com/p/kml-samples/issues/detail?id=207. However, it is unclear, if support will be added as Google Earth supports SVG only through WebKit and this makes it difficult to support SVG in icons (see https://groups.google.com/group/kml-support-getting-started/browse_thread/thread/ca7aefb14b2446f8).

This raises a more general issue, too, as the KML standard is unclear which formats may be referenced from an icon element. I.e., different KML clients may support different (and even non-overlapping) sets of image formats. If compliance testing is added for KML as part of CITE this most likely will lead to compliance testing issues. There seems to be an implicit assumption that the common bitmap formats (png, jpeg and gif) should be supported.

The issue was addressed by converting the symbol from SVG to PNG in the portrayal registry processing service.

7.2.4.3 Default styles in KML

Unlike in Symbology Encoding, a KML placemark will use a default icon and label style (yellow pushpin and the name of the placemark) unless these are overloaded for a style or are suppressed (e.g., using <IconStyle><scale>0</scale></IconStyle> and <LabelStyle><scale>0</scale></LabelStyle>). This needs to be taken into account when transforming Symbology Encoding symbolisers to KML.

7.2.4.4 Large ballons with minimal information

Since many attributes in the OWS-8 LTDS dataset contain null values in feature properties (encoded as "No Information", "noInformation", -999999, or -999999.0000000) the extended data shown in the balloons contains many entries and often it is not easy to spot the real values.

To reduce the information in the balloons to the properties that are not null, the encoding rule for the conversion has been amended to filter the LTDS-specific null value representations.

7.2.4.5 Scale

The portrayal rule set used in OWS-8 does not contain any scale ranges for portrayal rules (se:MinScaleDenominator and se:MaxScaleDenominator). As a result, all features are in principle visible in all scales, which is not optimal, in particular for data sets with features that are typically only drawn at large scales.

However, the testing showed that Google Earth still suppresses features on the map, if it "decides" that too much information would otherwise be drawn on the screen. As a result, this was not a real issue as the dataset was only a few MB in size. However, for a large dataset this might become an issue, as too much data is loaded (even if it is not drawn).

7.3 Improving the user experience by caching KML

To improve performance of the access to KML instances, a cache has been implemented on top of the Web Feature Service.

The cache provides a KML network link of the data in the Web Feature Service. The network link returns KML regions. Whenever a region is visible in the KML client in the appropriate level-of-detail, the client will send another request for the data in the region. The cache may return additional regions or KML placemarks. These placemarks are retrieved from the Web Feature Service, stored in the cache and returned to the KML client – unless the region is already stored in the cache. In this case, the cached data is returned directly without submitting another request to the cache.

This is illustrated in the figure below.

The lifespan of the data in the cache may be deleted after an expiry time, or if the portrayal rules and the KML transformation script of the WFS have been updated.

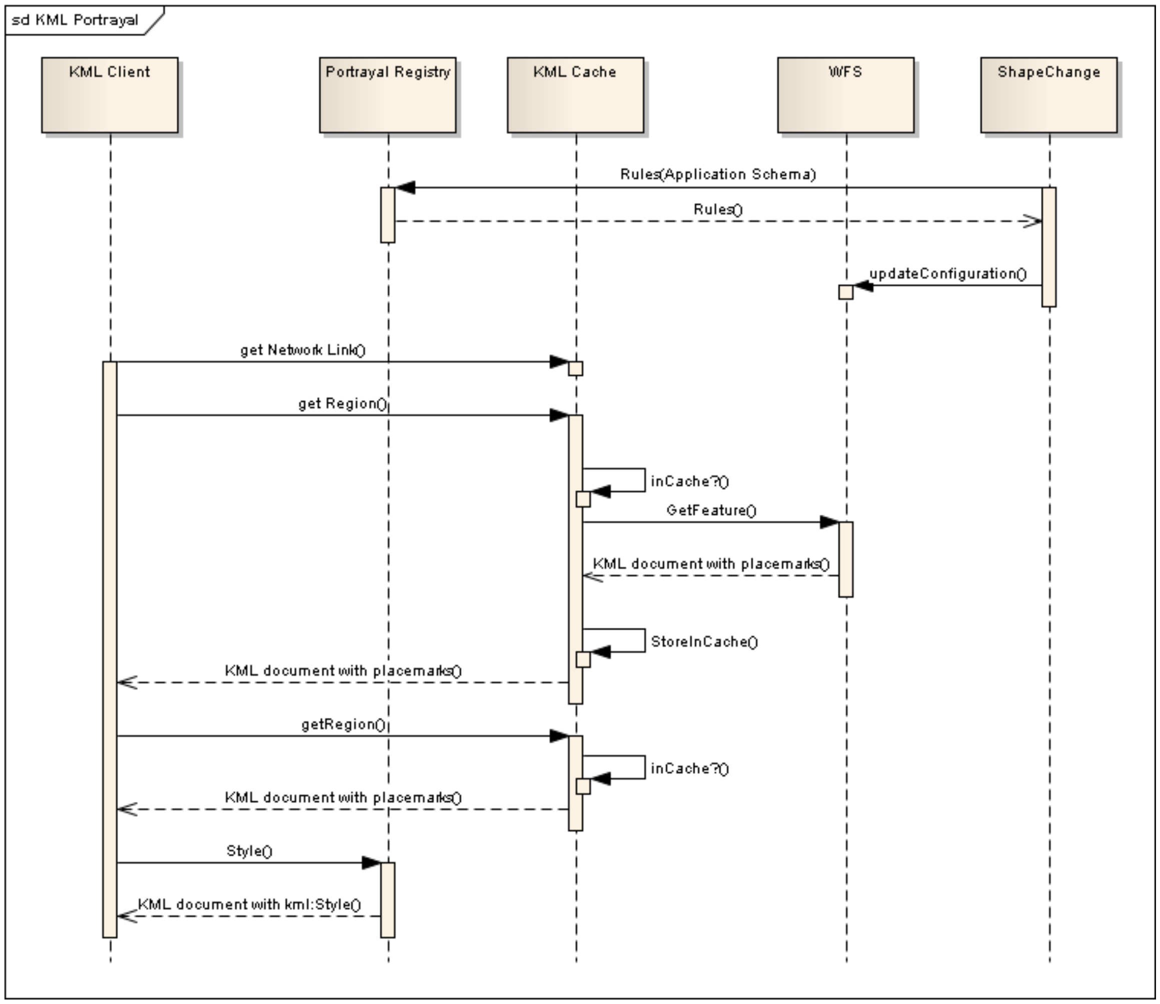


Figure 19 – Sequence diagram of the interaction between the KML client, the cache, the WFS and the portrayal registry

This approach is similar to the existing and effective caching mechanisms for Web Map Services (see, for example, GeoWebCache, MapProxy, or TileCache) and is relatively simple to implement, but effective.

However, while this reduce wait time for access to KML data, the Google Earth as a KML client becomes less smooth to work with, once a significant amount of KML data has been downloaded to the client (several MB) and all of this visible in the current view. I.e., it is essential to provide appropriate scale ranges with the KML data.

7.4 Using WFS 2.0 stored queries

To simplify access to the WFS, stored queries can be used to make it easier for clients to access relevant data. Let's use a simple example where a stored query is used to access all LTDS building features in a region. This can be extended to additional feature types by simply adding additional wfs:Query elements and including the feature type name in the returnedFeatureTypes attribute.

The following WFS operation creates the new query:

```
<wfs:CreateStoredQuery
   xmlns:wfs="http://www.opengis.net/wfs/2.0"
   xmlns:fes="http://www.opengis.org/fes/2.0"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:tds="http://metadata.dod.mil/mdr/ns/GSIP/3.0/tds/3.0"
   xmlns:xml="http://www.w3.org/XML/1998/namespace"
   xsi:schemaLocation="http://www.opengis.net/wfs/2.0
    http://schemas.opengis.net/wfs/2.0/wfs.xsd"
   service="WFS" version="2.0.0">
  <wfs:StoredQueryDefinition
   id="http://metadata.ces.mil/mdr/ns/GSIP/query/1">
    <wfs:Title xml:lang="en">LTDS features In a region</wfs:Title>
    <wfs:Abstract xml:lang="en">Sample WFS 2.0 stored query that can be
used to query all relevant TDS buildings in a region</wfs:Abstract>
    <wfs:Parameter name="LLat" type="double"/>
    <wfs:Parameter name="LLon" type="double"/>
    <wfs:Parameter name="ULat" type="double"/>
    <wfs:Parameter name="ULon" type="double"/>
    <wfs:QueryExpressionText
     returnFeatureTypes="tds:BuildingGeopoint tds:BuildingGeosurface"
     language="urn:ogc:def:queryLanguage:OGC-WFS::WFS_QueryExpression"
     isPrivate="false">
         <wfs:Query typeNames="tds:BuildingGeopoint">
            <fes:Filter>
             <fes:Within>
                <fes:ValueReference>tds:geometry</fes:ValueReference>
                <gml:Envelope srsName="http://metadata.ces.mil/mdr/ns
/GSIP/crs/WGS84E_2D">
                    <gml:lowerCorner>${LLat} ${LLon}</gml:lowerCorner>
                    <gml:upperCorner>${ULat} ${ULon}</gml:upperCorner>
                </gml:Envelope>
              </fes:Within>
            </fes:Filter>
         </wfs:Query>
         <wfs:Query typeNames="tds:BuildingGeosurface">
            <fes:Filter>
              <fes:Within>
                 <fes:ValueReference>tds:geometry</fes:ValueReference>
                <gml:Envelope srsName="http://metadata.ces.mil/mdr/ns
/GSIP/crs/WGS84E_2D">
                    <gml:lowerCorner>${LLat} ${LLon}</gml:lowerCorner>
                    <gml:upperCorner>${LLat} ${LLon}</gml:upperCorner>
                </gml:Envelope>
              </fes:Within>
            </fes:Filter>
         </wfs:Query>
      </wfs:QueryExpressionText>
   </wfs:StoredQueryDefinition>
</wfs:CreateStoredQuery>
```

Once the stored query is defined in the WFS, it the buildings in the bounding box with lower corner (36.5,-122) and upper corner (37,-121.5) can be accessed using the following query (special characters have not been escaped for better readability):

```
http://services.interactive-instruments.de/xsprojects/ows8-tds/cgi-
bin/ltds/wfs?
```

```
SERVICE=WFS&
VERSION=2.0.0&
REQUEST=GetFeature&
STOREDQUERY_ID=http://metadata.ces.mil/mdr/ns/GSIP/query/1&
LLat=36.5&
LLon=-122&
ULat=37&
ULon=-121.5
```

7.5 KML Change Request: Improve control over BalloonStyle layout

If one wishes to use JavaScript scripts in a KML Balloon it is currently required to provide the complete HTML document. This is usually undesirable, so it would be useful, if KML would support child elements to a BalloonStyle to specify, for example, styles and scripts.

More details on this topic can be found in the OWS-7 Schema Automation ER. As part of OWS-8, a change request on this issue has been submitted.

7.6 Conclusions

With a combination of caching KML regions and controlling the download of placemarks to the KML client using KML regions and level-of-detail information, access to large datasets from KML clients is possible using OGC Web Feature Services and simple extensions.

Using portrayal rules expressed using Symbology Encoding for KML portrayal has significant restrictions due to the different concepts used for portrayal in KML and Symbology Encoding. In general, the results will be satisfactory only for simple portrayal rules and symbolizers. Also, portrayal rules specified using Symbology Encoding do not support the interaction of the user in the KML client.

Annex A
(normative)

Conversion from OCL to Schematron

A.1 Translation principles

Translation from OCL to Schematron is performed on the basis of a ShapeChange-internal syntax representation of OCL expressions. The representation is close to the Concrete Syntax structure described in the OCL 2.2 standard [6].

Naturally, the syntax representation of OCL is recursive. Therefore the principles of translation from OCL to another language can best be described using a recursive notation. Below we describe, how some particular constructs (such as the application of the select() iterator:

x->select(t|pred(t)))

translate to XPath 1.0, where the translation results of the constituent parts (such as x and pred(t)) are presumed.

For a valid OCL expression x let τ(x) denote the equivalent XPath 1.0 expression. The expression x may contain free variables (explicit or implicit), which need to be treated when computing τ(x). One typical variable is *self*, which translates to *current().* So, τ(self)=current().

Note: The following table is a copy of the one prepared for OWS-7. It has been supplemented by the new constructs and extensions of OWS-8.

Category	OCL syntax	In words	Schematron translation
Variable access *self*	self	The current object in the context of which the expression shall hold.	current() Note: Whenever the current node happens to be identical to current() there is no need to explicitly generate current() for self. Relative path syntax is to be used in these cases.
Iterator variable access	t defined in an iterator used in x(t)	t has to be assigned a current value from the path that leads to x(t).	*If t has a realization in the path leading to x:* ../../.. …/.. As many .. as are required to reach the binding context of t. *No realization in the path (may be xlink:href):* Cannot be translated because there is no unique XPath expression to define this.
Let variable access	t defined in a let construct used in x(t)	t necessarily has a value from the	*If t is defined in the outer (current()) context:* $id, where id is some unique <let> variable. The let initializer is translated in the current()

Category	OCL syntax	In words	Schematron translation
		initialize expression.	context and initializes a Schemtron <let> element. *Other:* The let initializer of the variable t is translated in the current context and substitutes t.
Let expression	let x=y in z(x)	Assignment of expression y to variable x. Result is z(x).	*If x and y are defined in the outer (current()) context:* τ(z(τ(x))) Additionally, a Schematron <let> is created. *Other:* τ(z(τ(y))) This means we are substituting the initializers.
Integer or real constants	123 or 3.1415		same
Boolean constants	true or false		true() or false()
String constants	'xxxxx'		same
Enumeration constants	Type::value		'value'
Codelist constants	Type::value		*GML 3.3 rules:* The constant is translated to an external codelist reference according to a pattern in tagged values in the codelist class. *Other GML version rules:* 'value'
If expression	if x then y else z endif	If x evaluates to true then the value of the expression is y, otherwise z.	*If τ(y) and τ(z) are represented by nodesets:* τ(y)[τ(x)] | τ(z)[not(τ(x))] x needs to be compiled in the tail context of τ(y) and τ(z). *If τ(y) and τ(z) are strings:* concat(substring(τ(y),number(not(τ(x)))*string-length(τ(y))+1),substring(τ(z), number(τ(x))*string-length(τ(z))+1)) The trick is to concatenate substrings which either comprise the full argument or nothing, depending on the value of the predicate. *If τ(y) and τ(z) are numbers or Booleans:* As for strings. The result has to be converted into the proper type.

Attribute call	x . attname	Set of object instances reached from the	*If simple-typed (19136 encoding):*
			τ(x)/attname
			If simple-typed (19139 encoding – non-codelist):
			τ(x)/attname/*

Category	OCL syntax	In words	Schematron translation
		instance or set repre-sented by x by applying attribute *attname*.	*If simple-typed (19139 encoding – codelist):* τ(x)/attname/*/@codeListValue *If nested and complex-typed:* τ(x)/attname/* *If realized by means of xlink:href:* *[concat(α,@gml:id,β)=τ(x)/attname/@xlink:href] where α and β are constant prefixes and postfixes surrounding the identifier proper in the xlink:href value. The values for α and β can be configured. *7.6.1 If the type of linkage is unknown:* A nodeset union of the expressions above.
Attribute call according to nilReason implementation pattern	x . attname . value x . attname . reason	Set of instances reached by attname, respectively by attname/@nilReason	*Case x . attname . value:* τ(x.attname) Compilation as above – 'x.attname' is assumed to have the type of 'value'. *Case x . attname . reason (19136 encoding):* τ(x.attname)[@xsi.nil='true']/@nilReason *Case x . attname . reason (19139 encoding):* τ(x.attname)[not(*)]/@gco:nilReason
Operation call allInstances()	x . allInstances()	Set of all object instances of type x. x represents a type-valued expression.	*If x is a type constant:* Nodeset union (n_1|…|n_i), where n_k=//T_k[@gml:id] and T_k is one of the concrete derivations of the type of x (including x). *If x is a type expression:* Cannot be translated because required schema information is not available at run-time.
Operation call oclIsKindOf()	x . oclIsKindOf(y)	The single object instance x is checked for complying with type y.	*If y is a type constant:* boolean(τ(x)[name()='T_1' or … or name()='T_i']), where T_k is one of the names of the concrete derivations of y, including y. boolean(…) may be omitted if the argument is known to be used by operands, which do an implicit conversion to Boolean. *If y is a type expression:* Cannot be translated because required schema information is not available at run-time.
Operation call oclIsTypeOf()	x . oclIsTypeOf(y)	The single object instance x is checked for being of type y.	*If y is a type constant:* boolean(τ(x) [name()='T']), where T is the name of the type y. *If y is a type expression:* boolean(τ(x)/self::*[name()=name(τ(y))]) boolean(…) may be omitted if the argument is known to be used by operands, which do an implicit conversion to Boolean.

Category	OCL syntax	In words	Schematron translation
			Note: Expression part not implemented. *Type-comparing CharacterString to code lists:* We are making an exception to the strict rules with simple data elements which we permit being successfully type-compared to code lists.
Operation call oclAsType( )	x . oclAsType(y)	The single object instance x is downcast to type y. The value is 'undefined' if this is not possible.	*If y is a type constant:* τ(x)[name()='T_1' or … or name()='T_i'], where T_k is one of the names of the concrete derivations of y, including y. *If y is a type expression:* Cannot be translated because required schema information is not available at run-time. *Casting CharacterString to code lists:* We are making an exception to the strict rules with simple data elements which we permit being casted to code list types.

Operation call +,-,*,/	x + y, etc.	Value of x.+(y), etc.	τ(x) + τ(y)
			τ(x) - τ(y)
			τ(x) * τ(y)
			τ(x) div τ(y)

Operation calls =, <>	x = y,
	x <> y
		Value of x.=(y),
		x.<>(y)
			If x and y is are simple types:
			τ(x) = τ(y)
			τ(x) != τ(y)
			If x and y is are objects:
			generate-id(τ(x)) = generate-id(τ(y))
			generate-id(τ(x)) != generate-id(τ(y))

Operation call <, >, <=, >=	x < y	Value of x.<(y), etc.	τ(x) < τ(y)
			τ(x) > τ(y)
			τ(x) <= τ(y)
			τ(x) >= τ(y)

Operation call size()	x . size()	Number of characters in the string instance x.	string-length(τ(x))
Operation call concat()	x . concat(y)	String concatenation of x and y.	concat(τ(x),τ(y)) A series of concats may be joined to a multi-argument concat invocation.
Operation call substring()	x . substring(y,z)	Substring of x running from position y to position z	substring(τ(x), τ(y), τ(z)-τ(y)+1)

Operation call and, or, xor, implies	x and y
	x or y
	x xor y
		Logical combination as indicated	τ(x) and τ(y)
			τ(x) or τ(y)
			boolean(τ(x))!=boolean(τ(y))

Category	OCL syntax	In words	Schematron translation
	x implies y		not(τ(x)) or τ(y)
Set operation call size()	x -> size()	Number of objects in x.	count(τ(x))
Set operation call isEmpty()	x->isEmpty()	Predicate: Is the set represented by x empty?	not(τ(x))
Set operation call notEmpty()	x->notEmpty()	Predicate: Is the set represented by x not empty?	boolean(τ(x)) boolean may be omitted if τ(x) is known to be Boolean or is used by operands, which do an implicit conversion to Boolean.
Iterator call exists()	x -> exists(t|b(t))	Predicate: Does the set x contain an objects t for which the Boolean expression b(t) holds?	boolean(τ(x)[τ(b(.))]) boolean may be omitted if τ(x) is known to be Boolean or is used by operands, which do an implicit conversion to Boolean.

Iterator call forAll()	x -> forAll(t|b(t))	Predicate: Does the set x only contain objects t for which the Boolean expression b(t) holds?	count(τ(x))=count(τ(x)[τ(b(.))])
			
			In the implementation we map forAll() to exists(). We can do this because according to first level logic, we have:
			
			x->forAll(t|b(t)) = not(x->exists(t|not(b(t)))

Iterator call isUnique()	x -> isUnique(t|y(t))	Predicate: Does the set x only contain objects t for which the expression y(t) creates mutually different objects?	This is a hard one, which could only be solved in a few cases: *If y is a constant, y(t)=const:* count(τ(x))<=1 *If y is identity and x is object-valued, y(t)=t:* true() This is because nodesets are sets. *If y is identity and x is a collection of basic types, y(t)=t:* not(τ(x)[.=(preceding::*|ancestor::*)[count(.|τ(x))=count(τ(x))]]) This means any value in τ(x) must not be contained in the intersection of τ(x) with the previous part of the tree. *If y is an object-valued attribute, y(t)=t.a:* count(τ(x))=count(τ(x.a))

Category	OCL syntax	In words	Schematron translation
			This is true due to the pigeonhole principle. Note that t.a is required to be a single value, not a set! *If y is an attribute carrying a basic data type, y(t)=t.b (19136 encoding):* not(τ(x)[b=(preceding::*|ancestor::*)[count(.|τ(x))=count(τ(x))]/b]) This means the value of any b must not be contained in the intersection of τ(x) with the previous part of the tree. As above, t.b needs to be a single value. *If y is an attribute carrying a basic data type, y(t)=t.b (19139 encoding):* not(τ(x)[b/*=(preceding::*|ancestor::*)[count(.|τ(x))=count(τ(x))]/b/*]) Note: This is again different for 19139 codelist access. See “attribute call” row for this. *Nested attributes of either kind, y(t)=t.a1.a2…b:* Each single step needs to be unique. Hence we can reduce this to: τ(x->isUnique(t|t.a1)) and τ(x.a1->isUnique(t|t.a2)) and … and τ(x.a1.a2…->isUnique(t|t.b)) *Any other, particularly arbitrary expressions:* Cannot be translated because no way to express this in XPath 1.0 has been found.
Iterator call select()	x -> select(t|b(t))	Compute the set of those objects t in x, for which the predicate b(t) holds.	τ(x) [τ(b(.))] Note that this is very similar to exists(), the only difference being the Boolean interpretation of the result in the exists() case.
Pattern matching function on Strings	x . matches(pattern) Note: This operation call is an extension. It is not part of the OCL standard.	Boolean function which yields true if the pattern of type String matches the String argument.	There is no way to express matches() in XPath 1.0 except by way of using a Java extension function or by making use of the matches function available in XPath 2.0. The implementation allows configuring either the use of an extension function or of XPath 2.0 syntax. The XPath translation target is configurable text (a function call), which receives τ(x) and τ(pattern) as substitutes for the strings ‘$object$’ and ‘$pattern$’, which both have to be part of the configured function

Category	OCL syntax	In words	Schematron translation
			call.

Annex B (normative) KML encoding rule

B.1 General encoding requirements

B.1.1 Application schemas

The application schema shall conform to the same UML profile specified in GML 3.2 Annex E with the following additional, optional tagged values:

Table B.1 — Tagged values

UML model element	Tagged value	Description
Package	kmlStyleUrl	an absolute URL referencing a kml:Style or kml:StyleMap element
Class with stereotype <<featureType>>	name	a human readable name of the feature type
	description	a human readable description of the feature type
	kmlReference	an absolute URL to a resource that includes a description of the feature type
	kmlStyleUrl	an absolute URL referencing a kml:Style or kml:StyleMap element
Attribute or association end	name	a human readable name of the property type
	description	a human readable description of the property type
	kmlReference	an absolute URL to a resource that includes a description of the property type
Attribute	kmlName	if the value is set to “true”, and an instance has a value for this property, that value is used as the name of the placemark instance in KML
	kmlTimeSpanBegin	if the value is set to “true”, and an instance has a value for this property, that value is used as the begin of the time span element of the placemark instance in KML
	kmlTimeSpanEnd	if the value is set to “true”, and an instance has a value for this property, that value is used as the end of the time span element of the placemark instance in KML
	kmlTimeStamp	if the value is set to “true”, and an instance has a value for this property, that value is used as the time stamp element of the placemark instance in KML

For the four last tagged values listed in table A.1 the following rule applies, if multiple property types of a feature type and its super-types are tagged in this way: The property types of the feature type itself are inspected. If one or more property types contain the tagged value with a value of “true”, a property type is selected randomly. If none of the property types carries such a tagged value, the process is continued per super-type recursively until either a property type with the tagged value is found or all property types have been inspected.

In addition, the values of spatial properties in the application schema shall conform with version 1.2 of the OGC standard “Simple feature access - Part 1 - Common architecture”.

NOTE KML only supports linear interpolations and no sharing of geometries between features, like the simple feature access standard.

B.1.2 Character repertoire and languages

“UTF-8” or “UTF-16” shall be used as the character encoding of all XML files (with the associated character repertoire).

B.1.3 Exchange metadata

No specific rules for exchange metadata is specified by this encoding rule.

B.1.4 Dataset and object identification

Unique identifiers in accordance with XML's ID mechanism are used to identify elements.

NOTE The XML ID mechanism only requires that these identifiers are unique identifiers within the XML document in which they appear.

B.1.5 Update mechanism

The general KML mechanisms for updates apply.

B.2 Input data structure

See ISO/DIS 19118, Clause 8, for a description of the input data structure.

B.3 Output data structure

See KML 2.2 for a description of the output data structure.

NOTE In this encoding rule the namespace prefix “kml” refers to the namespace of KML, which is “http://www.opengis.net/kml/2.2”.

B.4 Conversion rules

B.4.1 Instance conversion rules

In general, the conversion rules use data conforming to the application schema and structured according to the generic instance model (see ISO/DIS 19118, Clause 8). However, the following description is based on the instance model of the GML representation of the data as this is a better known representation of the data than the generic instance model. If required, conversion rules using the generic instance model might be added in a future revision.

The converted instances shall be represented in a KML document.

Every feature instance ("//schema-element(gml:AbstractFeature)") shall be represented as a kml:Placemark element.

The placemark element shall have the following child nodes:

- An attribute "id" with the value of "@gml:id".
- An element "kml:name" with the following value:
    - If a property type of the feature type has a tagged value "kmlName" with a value of "true" and the feature instance has a value for this property, this value is used. If multiple name property values exist, only the first one is selected.
    - Alternatively, if the feature instance has a "gml:name" value, this value is used. If multiple name property values exist, only the first one is selected.
    - Alternatively, if the feature type has a tagged value "name", its value is used.
    - As a fallback, the name of the feature type is used.
- An element "kml:visibility" with a value of "1".
- An element "kml:styleUrl" with the following value:
    - If the feature type has a tagged value "kmlStyleUrl", this value is used.
    - Alternatively, if the package containing the feature type has such a tagged value, this value is used.
    - Alternatively, if the package containing that has such a tagged value and is within the same application schema, this value is used. This is applied recursively.
    - Alternatively, if the ShapeChange configuration includes a reference to a portrayal rule set, the rule set is retrieved and all portrayal rules for the feature type in the rule set are collected and processed in reverse order (due to the painters model). Each filter expression of a rule is converted into an Xpath expression so that the first "firing" expression will result in that the value of the se:OnlineResource of the rule is used as the value.
    - If no value is found, the "kml:StypeUrl" element is omitted from the placemark instance.

- An element “kml:TimeSpan/kml:begin” with the following value:
  - If a property type of the feature type has a tagged value “kmlTimeSpanBegin” with a value of “true” and the feature instance has a value for this property, this value is used. If multiple name property values exist, only the first one is selected.
  - If no such value is found, the element is omitted from the placemark instance.
- An element “kml:TimeSpan/kml:end” with the following value:
  - If a property type of the feature type has a tagged value “kmlTimeSpanEnd” with a value of “true” and the feature instance has a value for this property, this value is used. If multiple name property values exist, only the first one is selected.
  - If no such value is found, the element is omitted from the placemark instance.
- An element “kml:TimeStamp/kml:when” with the following value:
  - If a property type of the feature type has a tagged value “kmlTimeStamp” with a value of “true” and the feature instance has a value for this property, this value is used. If multiple name property values exist, only the first one is selected.
  - If no such value is found, the element is omitted from the placemark instance.
- An element “kml:ExtendedData” with a child element “kml:SchemaData”. That element shall have an attribute “schemaUrl” with the value that is the concatenation of “#”, the name of the feature type, and “Schema”.
- A child element “kml:SimpleData” with
  - an attribute “name” and a value of “type"
  - a CDATA block[4]

  The CDATA block contains the following value:
  - If the feature instance has a “gml:description” value, this value is used.

[4] Whenever the encoding rule requires that a CDATA block is used in the KML instance document, an XML processor may encode the CDATA block in an equivalent way, i.e. with “<” encoded as “&lt;” and “>” encoded as “&gt;”.

  - o Alternatively, if the feature type has a tagged value “description”, its value is used.
  - o Alternatively, the documentation of the feature type is used.

- For each property element of the feature that is not “gml:description”, “gml:name”, or a geometry (“*/schema-element(gml:AbstractGeometry)”) and which is not nil, another child element “kml:SimpleData” is added with

  - an attribute “name” and the local name of the property as the value
  - a CDATA block

  The value of the CDATA block depends on the value type of the property:

  - a number, string or enumeration type: the value (“.”).
  - a date or time type: the value (“.”) styled in a human readable form.
  - a measure: the value (“.”) plus the value of “@uom”.
  - a code list value: If the value has a codeSpace attribute the CDATA block contains “<a href=’$[codespace]’>$[value]</a>” where “$[codespace]” is the code space attribute value and “$[value]” is the value of the property element. Otherwise, the CDATA block contains just “$[value]”.
  - a structured data type: a CDATA block with a table that has a row for each property element of the structured data type that is not nil. The left column contains the local name of the property element “<small><i>$[local-name]</i></small>“. If the property element has no child elements, the right column contains the value (“.”), otherwise the value is treated as another structured data type and contains a nested table in accordance with the conversion rules in this paragraph.

    NOTE 1 This conversion rule results in a good representation in the most common cases, where the nesting is not deep and values are generally of simple content. However, as it is easy to construct cases where this rule will not result in a satisfactory result, this rule may require some refinement after more experience with a range of application schemas.

  - a feature that is referenced by an Xlink: A CDATA block “<a href=’$[ref]’>$[value]</a>” where “$[ref]” is the value of “@xlink:href”. If “$[ref]” contains a “#” then “$[value]” is the string after the “#”, otherwise the value is just “Reference”.
  - any other type: the value (“.”).

  NOTE Support for additional types will be required depending on the use of types in application schemas and will be added to this encoding rule as needed.

Multiple values of the same property are separated by "<hr/>".

- a "kml:Point" or "kml:MultiGeometry" depending on the spatial property:
    - a "gml:Point" is converted to a "kml:Point"
    - a "gml:LineString" or "gml:Curve" is converted to a "kml:MultiGeometry" with a "kml:Point" and a "kml:LineString". The point is one of the control points of the curve. It is recommended to select a control point in the middle of the curve.
    - a "gml:Polygon" or "gml:Surface" is converted to a "kml:MultiGeometry" with a "kml:Point" and a "kml:Polygon". The point is the centroid of the polygon.
    - a "gml:MultiGeometry" is converted to a "kml:MultiGeometry".

  NOTE 3 The additional points for curves and surfaces are required for icons and labels, if included in the style definition.

All coordinates shall be transformed to WGS84, if required.

NOTE 4 KML uses coordinate order long/lat, so coordinates in CRS urn:ogc:def:crs:EPSG::4326 or urn:ogc:def:crs:EPSG::4979 need to be converted with a different axis order.

All "kml:Schema" elements created by applying the Schema conversion rules shall also be added to the KML document.

B.4.2 Schema conversion rules

The schema conversion rules define how reusable KML fragments shall be derived from an application schema expressed in UML in accordance with ISO 19109. A number of general rules are defined in A.2.4 to describe the mapping from a UML model that follows the guidelines described in A.2.1.

Every feature type in the application schema shall be represented as a kml:Schema element. The element shall have the following child nodes:

- An attribute "id" with the local name of the feature type + "Schema".
- A child element "kml:SimpleField" with
    - an attribute "name" and a value of "type"
    - an attribute "type" and a value of "string"
    - an element "displayName" with a CDATA block with the following value:

- If the feature type has a tagged value “kmlReference” the CDATA block contains “<a href=’$[kmlReference]’ title=’$[documentation]’><big><b><i>$[type]</i></b> </big></a>” where “$[kmlReference]” is the value of the tagged value, “$[documentation]” the documentation of the feature type and “$[type]” is the local, if available human-readable, name of the feature type. Otherwise, the CDATA block contains “<div title=’$[documentation]’><big><b><i>$[type]</i></b> </big></div>”.

- For each property type of the feature that does not have a local name “description” or “name” or is a geometry, another child element “kml:SimpleField” is added with
  - an attribute “name” and the local name of the property type as the value
  - an attribute “type” and a value of “string”
  - an element “displayName” with a CDATA block; if the property type has a tagged value “kmlReference” the CDATA block contains “<a href=’$[kmlReference]’ title=’$[documentation]’>$[property]</a>” where “$[kmlReference]” is the value of the tagged value, “$[documentation]” the documentation of the property type and “$[property]” is the local name of the property type. Otherwise, the CDATA block contains “<div title=’$[documentation]’><b><i>$[property]</i></b></div>”.

Bibliography

[1] OGC® OWS-5 GSIP Schema Processing Engineering Report, OGC document 08-078r1

[2] OGC® OWS-6 GML Profile Validation Tool Engineering Report, OGC document 09-038r1

[3] OGC® OWS-7 Schema Automation Engineering Report, OGC document 10-088r2

[4] OGC® The Specification Model — Modular specifications, OGC document 08-131r3

[5] OGC® Web Feature Service, version 2.0.0, OGC document 09-025r1 (OGC standard)

[6] Object Constraint Language, Version 2.2, OMG Object Management Group