Open Geospatial Consortium

Publication Date: 2014-04-15

Approval Date: 2014-03-30

Submission Date: 2012-10-1

External identifier of this OGC® document: http://www.opengis.net/doc/er/ows7/schema-automation

Reference number of this document: OGC 10-088r3

Category: Engineering Report

Editor: Clemens Portele

OGC® OWS-7 Schema Automation Engineering Report

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. This document is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

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

License Agreement

Abstract

The capabilities of OGC's KML 2.2 as a format for exchange and visualization of U.S. National System for Geospatial Intelligence (NSG) Application Schema (NAS) data is explored.

Keywords

ogcdoc, ows7, nas, kml

Preface

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

Forward

Contents

Page

OGC® OWS-7 Schema Automation Engineering Report

1 Introduction

1.1 Scope

One work item of the OWS-7 initiative was to further the development and interoperability of application schemas. Using the U.S. National System for Geospatial Intelligence (NSG) Application Schema (NAS) as a case study two improvements to the automation of schema development have been addressed:

- The capabilities of OGC's KML 2.2 as a format for exchange and visualization of NAS data were explored. This involved several sub-topics:
    - exploring how NAS-conformant data could be represented in KML to achieve an appropriate visualization in KML clients, in particular Google Earth;
    - defining a KML encoding rule for a UML application schema and its accompanying GML-based schema to allow for an automated conversion of GML instance data into KML data;
    - enhancing ShapeChange by implementing the KML encoding rule as an XSLT stylesheet that can be applied to GML instance documents of the same application schema to generate a KML document;
    - testing the approach with NAS-conformant data accessed via the WFS interface as either GML 3.2 or KML 2.2 data
- OWS-5 and OWS-6 have started to investigate the use of Schematron to represent OCL constraints from the application schema to enable the validation of GML instance data beyond the syntactic validation possible using XML Schema. It was found that the general approach was feasible and useful. Building upon these results, OWS-7 aimed at a more comprehensive approach. The following items were addressed:
    - OCL 2.2 constraints have been added to the NAS UML model;
    - comprehensive rules for the conversion of OCL constraints to Schematron assertions have been developed;
    - these conversion rules have been implemented in ShapeChange;

- the conversion rules have been tested using the enriched UML model of the NAS.

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.

1.2 Document contributor contact points

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

Name	Organization
Clemens Portele (editor)	interactive instruments GmbH
Reinhard Erstling	interactive instruments GmbH
Remi Koblenzer	interactive instruments GmbH
Dave Wesloh	NGA
Paul Birkel	MITRE

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2010-05-27	0.0.1	C. Portele	all	
2010-06-10	0.0.2	C. Portele P. Birkel R. Erstling	Future work 5.3 5.4 5.5 Annex B	
2010-10-21	0.0.3	R. Erstling	1.4, 6.4, 6.5, 6.6, 6.7, B.1, B.4	

1.4 Future work

The following items for future work have been identified:

- Display of tooltips: 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. See 5.2.2.

- Encoding rule improvements to support fine-grained styles. See 5.3.2.
- The role of portrayal registries for both KML and SLD needs further investigation. See 5.3.2.
- Investigation of performance improvements: The encoding of KML data directly from the source data without generating GML as an intermediate step as well as the caching of data in KMZ files. See 5.5.
- In the OCL-to-Schematron translation the treatment of Xlink-based references may be improved. As currently implemented, the reference targets have to be contained in the document being tested. It would be useful to investigate, how reference targets in external documents might be drawn on. See 6.7.1 for a full discussion.
- There are language elements in OCL, which are not really necessary, but greatly increase the comfort in use. These have not been covered in the implementation. The *let* construct of OCL (see 6.7.2) and many operations and iterators of OCL's build-in types belong to this class.
- Full support for ISO 19139 encoding rules should be added to the OCL-to-Schematron translation to support embedded metadata. See 6.7.3.
- There had been a discussion on how to best represent nillable data, which has cardinality > 1. See 6.7.4 for a full discussion.
- The XPath 1.0 language, which is the basis of the ISO Schematron standard, has a rather limited functionality, which sets hard limits on the OCL, which can effectively be translated. XPath 2.0 does not have these limitations, but is currently not part of Schematron. See 6.7.5 for that.

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)

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

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

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
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.

5 KML as a format for exchange and visualization of NAS data

5.1 Identifying the appropriate representation of NAS data in KML

5.1.1 Encoding Options

5.1.1.1 Overview

KML 2.2 offers the kml:ExtendedData element that enables the inclusion of additional data elements in a KML placemark. The kml:ExtendedData element offers three mechanisms for adding user-defined data to a feature. These mechanisms are:

- adding arbitrary untyped name/value data pairs using the kml:Data element

- adding instances of typed fields defined in the user-defined kml:Schema element, plus additional elements in some other namespace

- including any XML content defined in namespaces other than the KML namespace

A simple KML file with three polygon features was used to test which extended data is presented in clients - and how. Each option was encoded in one feature as discussed below.

5.1.1.2 Building-1: Using kml:Data

In this example, five data elements are included in the ExtendedData element. These are just name-value-pairs without any differentiation of types. An optional display name can be provided with HTML tag to show how the data element should be presented. The use of the CDATA wrapper ensures that HTML formatting elements are preserved for use by the client inside of the KML placemark “balloon” presentation.

```
<ExtendedData>
    <Data name="type">
        <displayName><![CDATA[<b>Type</b>]]></displayName>
        <value>
            <![CDATA[
            <a href="https://www.dgiwg.org/FAD/fdd/view?i=111415">
            Building
            </a>]]>
        </value>
    </Data>
    <Data name="height">
        <displayName><![CDATA[<b>Height</b>]]></displayName>
        <value>54 ft</value>
    </Data>
    <Data name="stories">
        <displayName><![CDATA[<b>Stories</b>]]></displayName>
        <value>3</value>
    </Data>
    <Data name="building type">
        <displayName>
            <![CDATA[
            <b>
                <a href="https://www.dgiwg.org/FAD/fdd/view?i=100853">
                Educational facility type
                </a>
            </b>
            ]]>
        </displayName>
        <value>
            <![CDATA[
            <a href="https://www.dgiwg.org/FAD/fdd/view?i=103456">School</a>
            <a href="#b3">Building-3</a>
            ]]>
        </value>
    </Data>
</ExtendedData>
```

Advantages:

- no schema required
- easy to parse
- display information available for KML clients (CDATA-wrapped)

Disadvantages:

- only simple types
- display names have to be repeated for every feature instance
- no type information

5.1.1.3 Building-2: Test feature using kml:Schema and kml:SchemaData

Similar four data elements to the ones in Bulding-1 above have been specified, in addition some elements which are in the substitution group of kml:SchemaDataExtension (in the instance) or kml:SchemaExtension (in the schema description) are added. These are just name-value-pairs without any differentiation of types. An optional display name can be provided with HTML tag to show how the data element should be presented.

```
<ExtendedData>
    <SchemaData schemaUrl="#BuildingTypeId"
        xmlns:test="http://www.opengis.net/ows7/test">
        <SimpleData name="type">
            <![CDATA[
            <a href="https://www.dgiwg.org/FAD/fdd/view?i=111415">Building</a>
            ]]>
        </SimpleData>
        <SimpleData name="height">54 ft</SimpleData>
        <SimpleData name="stories">3</SimpleData>
        <SimpleData name="building type">
            <![CDATA[
            <a href="https://www.dgiwg.org/FAD/fdd/view?i=103456">School</a>
            <a href="#b3">Building-3</a>
            ]]>
        </SimpleData>
        <test:element1>Test</test:element1>
        <test:element2 attribut="test"/>
        <test:element3>
            <test:element4 uom="m">45.2</test:element4>
        </test:element3>
        <test:ref>http://www.opengeospatial.org/</test:ref>
    </SchemaData>
</ExtendedData>
```

where the schema referenced from schemaUrl is specified as

```
<Schema name="Building" id="BuildingTypeId"
     xmlns:test="http://www.opengis.net/ows7/test">
    <SimpleField type="string" name="type">
        <displayName><![CDATA[<b>Type</b>]]></displayName>
    </SimpleField>
    <SimpleField type="string" name="height">
        <displayName><![CDATA[<b>Height</b>]]></displayName>
    </SimpleField>
    <SimpleField type="double" name="stories">
        <displayName><![CDATA[<b>Stories</b>]]></displayName>
    </SimpleField>
    <SimpleField type="string" name="building type">
        <displayName>
            <![CDATA[
            <b><a  href="https://www.dgiwg.org/FAD/fdd/view?i=100853">Educational
        facility type</a></b>
            ]]>
        </displayName>
    </SimpleField>
    <test:TestField name="element1"/>
    <test:TestField name="element2"/>
    <test:TestField name="element3"/>
    <test:TestField name="ref"/>
</Schema>
```

Advantages:

- no external schema required
- display information only needs to be provided once (in kml:Schema)
- easy to parse
- display information available for KML clients (CDATA-wrapped)
- can be extended

Disadvantages:

- only simple types
- limited type information

5.1.1.4 Building-3: Test feature using predefined elements

Finally, we encoded XML elements from an existing schema in an instance:

```
<ExtendedData xmlns:test="http://www.opengis.net/ows7/test">
    <test:element1>Test</test:element1>
    <test:element2 attribute="test"/>
    <test:element3>
        <test:element4 uom="m">45.2</test:element4>
        <test:element5>some text</test:element5>
    </test:element3>
    <test:ref>http://www.opengeospatial.org/</test:ref>
</ExtendedData>
```

Advantages:

- existing schema can be reused
- structured can be encoded properly

Disadvantages:

- requires additional schema
- no generic display information available for KML clients

5.1.2 Behaviour in KML clients

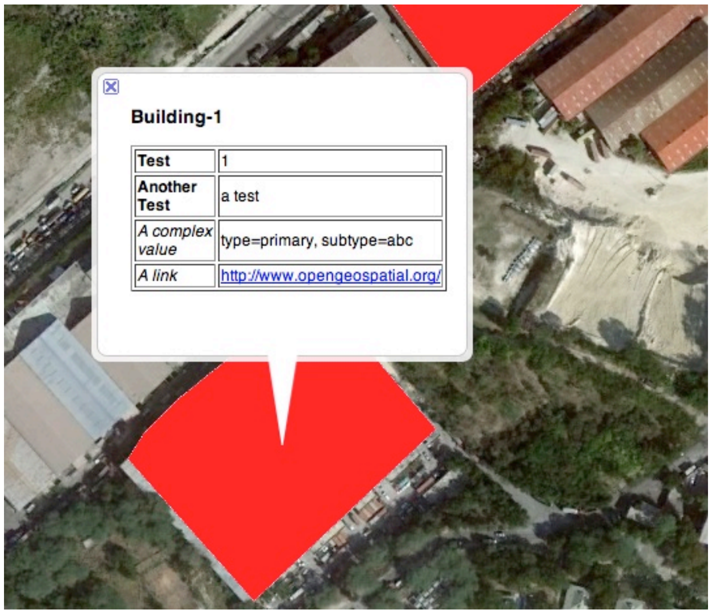
5.1.2.1 Overview

This section illustrates how the different kinds of extended data sections are visualised in selected KML clients.

5.1.2.2 Google Earth

Google Earth displays all information that is in elements in the KML namespace (kml:Data and kml:Schema/kml:SimpleData). It uses the displayName information. All information in other namespaces is ignored.

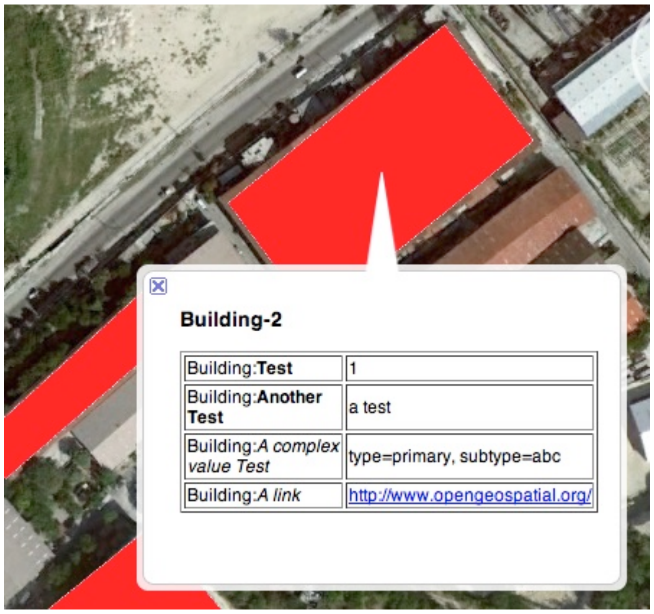
Building-1:



Building-1

Test	1
Another Test	a test
A complex value	type=primary, subtype=abc
A link	http://www.opengeospatial.org/

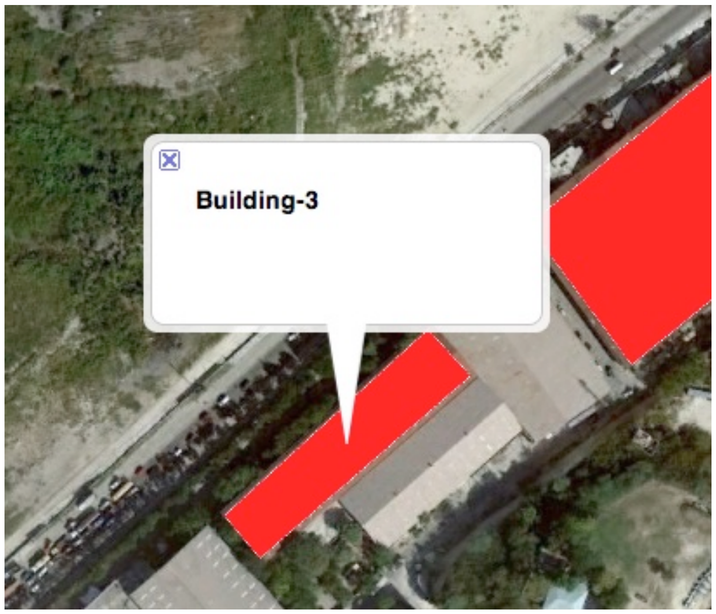
Building-2:



Building-2

Building:**Test**	1
Building:**Another Test**	a test
Building:*A complex value Test*	type=primary, subtype=abc
Building:*A link*	http://www.opengeospatial.org/

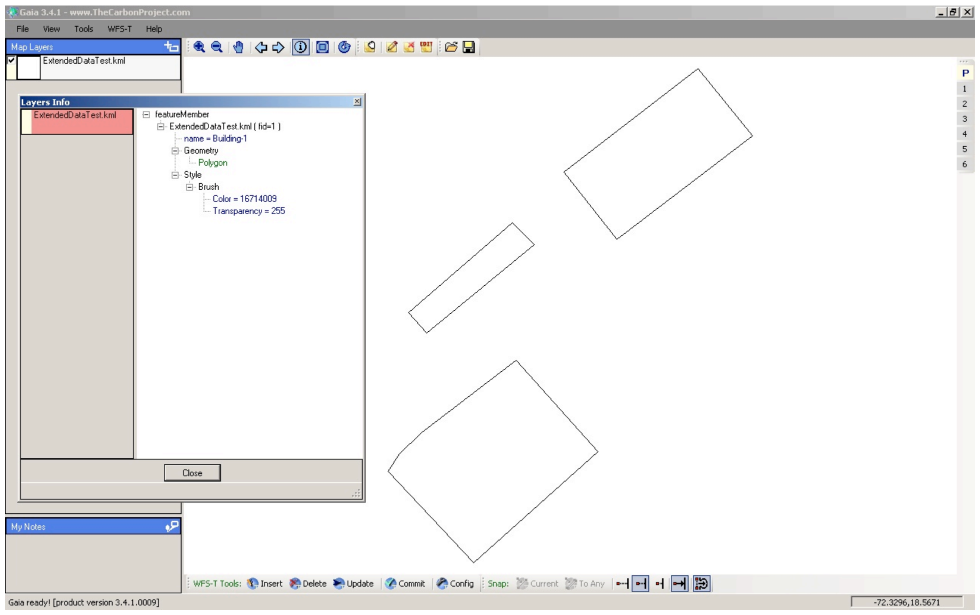
Building-3:



5.1.2.3 Gaia

Gaia ignores the extended data information including in its info tool. Gaia also seems to ignore the style information.

Building-1:

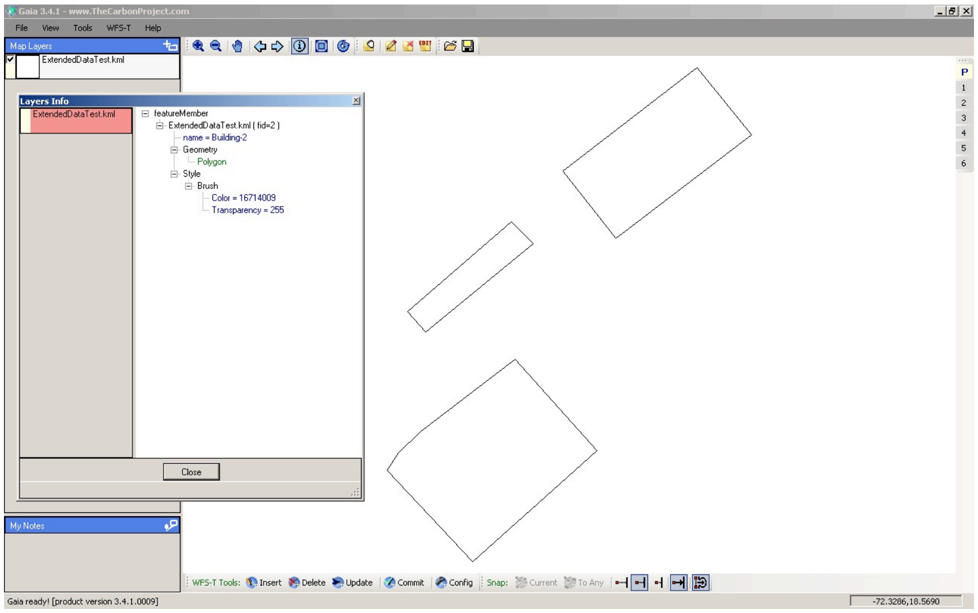

Gaia 3.4.1 - www.TheCarbonProject.com
File View Tools WFS-T Help
Map Layers
ExtendedDataTest.kml
Layers Info
ExtendedDataTest.kml
Close
My Notes
WFS-T Tools: Insert Delete Update Commit Config
Snap: Current To Any
P
1
2
3
4
5
6

```
featureMember
  ExtendedDataTest.kml ( fid=1 )
    name = Building-1
    Geometry
      Polygon
    Style
      Brush
        Color = 16714009
        Transparency = 255
```

Gaia ready! [product version 3.4.1.0009]

-72.3296,18.5671

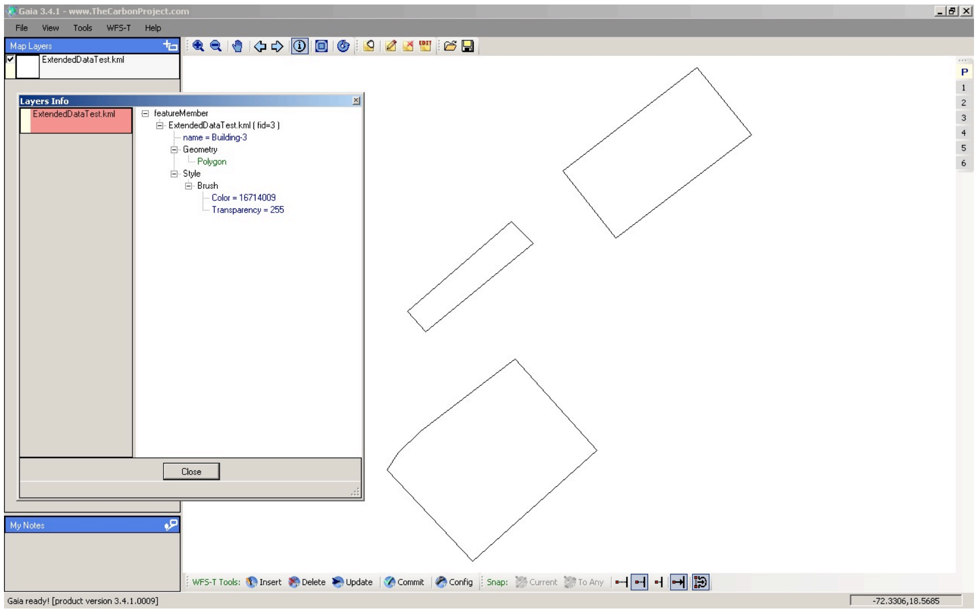
Building-2:


Gaia 3.4.1 - www.TheCarbonProject.com
File View Tools WFS-T Help
Map Layers
ExtendedDataTest.kml
Layers Info
Close
My Notes
WFS-T Tools: Insert Delete Update Commit Config Snap: Current To Any

```
featureMember
  ExtendedDataTest.kml ( fid=2 )
    name = Building-2
    Geometry
      Polygon
    Style
      Brush
        Color = 16714009
        Transparency = 255
```

Gaia ready! [product version 3.4.1.0009] -72.3286,18.5690

Building-3:


Gaia 3.4.1 - www.TheCarbonProject.com
File View Tools WFS-T Help
Map Layers
ExtendedDataTest.kml
Layers Info
Close
My Notes
WFS-T Tools: Insert Delete Update Commit Config Snap: Current To Any

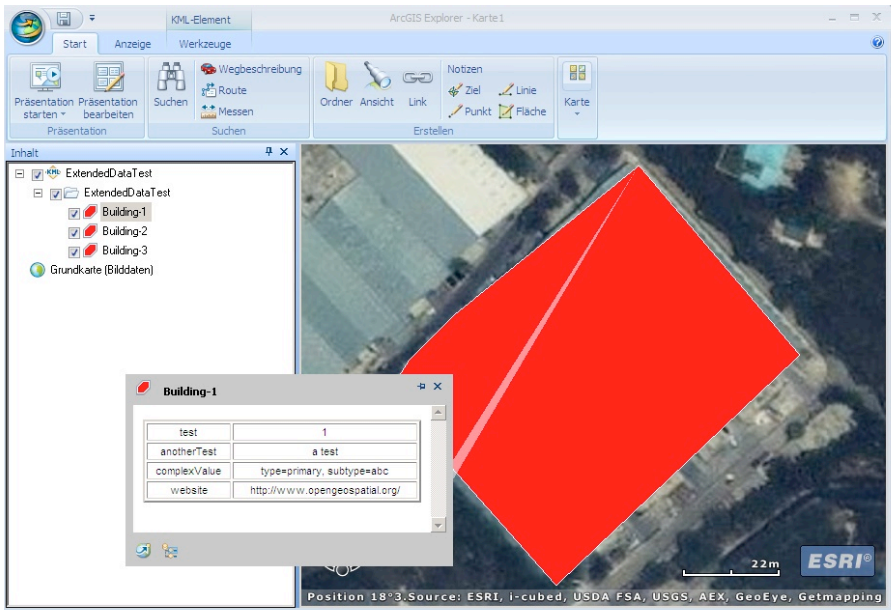
```
featureMember
  ExtendedDataTest.kml ( fid=3 )
    name = Building-3
    Geometry
      Polygon
    Style
      Brush
        Color = 16714009
        Transparency = 255
```

Gaia ready! [product version 3.4.1.0009] -72.3306,18.5685

5.1.2.4 ArcGIS Explorer

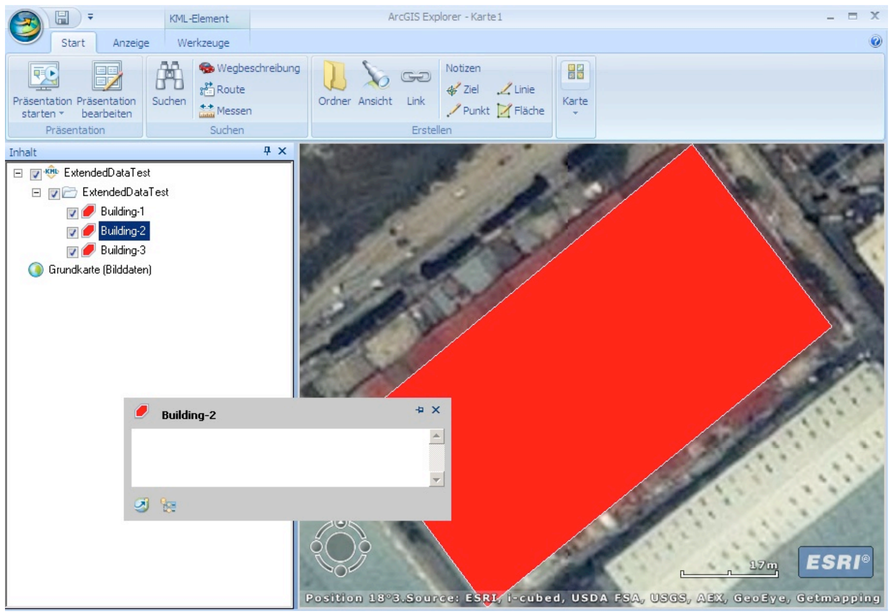
ArcGIS Explorer uses the style information and displays kml:Data elements (@name and value pairs in a table). Other extended data options are not displayed.

Building-1:

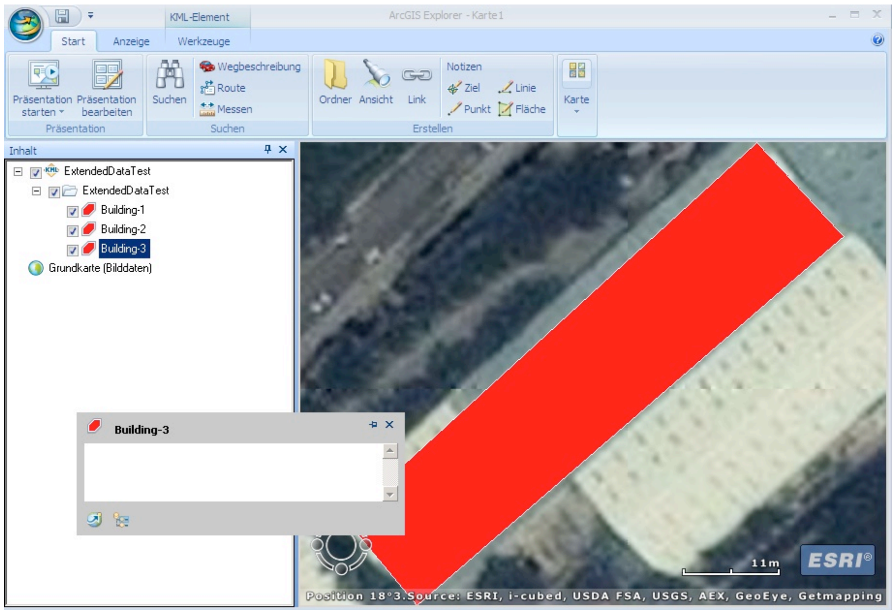


test	1
anotherTest	a test
complexValue	type=primary, subtype=abc
website	http://www.opengeospatial.org/

Building-2:



Building-3:



5.1.3 Conclusions

Use of XML elements in other namespaces is not supported by any existing client and should be avoided for the encoding of UML application schema (e.g., NAS-conformant) instance data as it is the goal that the data should be viewable in standard KML clients, in particular Google Earth.

The advantage of kml:SchemaData over kml:Data is less repetition of styling information (relevant if a significant number of features with a lot of extended data is encoded).

5.2 KML encoding rule

5.2.1 General concepts

Based to conclusions of the tests above, the KML encoding rule in Annex A has been specified.

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 A 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 will not make full use of the capabilities of KML.

The rules for the instance conversion, as currently documented in Annex A, are worded so that GML data is assumed as input. This reflects the planned use of KML as output 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 and additional information is represented in kml:ExtendedData elements.

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.

5.2.2 Display of tooltips

There are differences in how balloons are styled in different clients and also how the same client behaves on different operating systems.

A particular issue that was recognized during testing was that tooltips are rendered differently in Google Earth 5 on OS X and on Windows – even though WebKit is used as the rendering engine on both platforms. The behaviour in most current web browsers seems to be to preserve blanks and line breaks (e.g. in Chrome on OS X and Windows, Safari on OS X, IE on Windows); only Firefox on OS X preserved blanks, but no line breaks. On OS X, blanks and line breaks are preserved, but not on Windows, which makes longer documentation fields hard to read.

A possibility to overcome this issue would be creating custom tooltips. This would require the use of JavaScript and CSS. This is possible in KML in general, but would require that the complete HTML document shown in the balloon is specified in the BalloonStyle element in the referenced style. I.e., this would require that the tables which a rendered from the elements in kml:ExtendedData would have to be explicitly processed in the BalloonStyle, which has severe disadvantages:

- More custom code to create and manage, less out-of-the-box behaviour (and no automatic benefit from further development of KML client software as its behaviour would be overridden).
- All styles would become type dependent and would have to be created and managed as part of the encoding rule. This would make the entire process more fragile and also would require that other styling information like colours become part of the UML model - something that seems counterproductive.
- The typical default behaviour just omits rows where the KML instance has no value. This is not so easy to accomplish, if the encoding rule would have to create complete balloon HTML documents.

Therefore, this has not been considered an option.

Another approach would be a (future) KML extension that would add child elements to a BalloonStyle for styles and scripts.

5.3 Role of styles

5.3.1 Styles in the context of OGC Portrayal Services

OGC Portrayal Services rely on a portrayal registry where portrayal rules are represented using SLD & SE. These are accessed by web services implementing the OGC WMS/SLD interface.

There is no standard for the service interface to access items in the portrayal registry. In the aviation thread of OWS-7 the CSW ebRIM interface was used, but any other interface or implementation could probably be used as well as long as it supports the following requirements:

- Definitions of layers and their styles can be requested in a <StyledLayerDescriptor> representation as specified by WMS/SLD.
- A set of layers and styles required in a single GetMap request can be accessed from the registry using a single URL.
- Once published, such URLs must remain active forever - or at least as long as anyone might reference the particular set of layers/styles.

5.3.2 Styles and portrayal registries in the context of KML data

KML does not have a concept of layers and styles associated with these layers. Instead, each feature references the style in which it should be rendered.

This results in a different way how styles work in SLD/SE and KML. In SLD/SE a filter identifies the features of a feature type to which a specific style applies (data and portrayal are separated). In KML, however, the applicable style is part of the feature itself. In the encoding rule in Annex A a relatively simple mechanism is supported: all features of a particular feature type are rendered using the same style. This may be too simple in many cases and might require a more refined mechanism in the future, e.g. to distinguish in the symbology different building functions or conditions.

As rendering of KML data usually is done by the client (and not the server as in the case of an OGC Portrayal Service) there are also functional differences. In particular, KML styling natively supports a different rendering of features that are selected/highlighted by the user.

In principle, the requirements on a portrayal registry supporting KML data are similar to a portrayal registry supporting OGC Portrayal Services:

- Definitions of styles can be requested in a <kml> representation with <Style> and/or <StyleMap> child elements as specified by KML.

- Styles can be accessed from the registry using a URL with a fragment identifier to reference the particular Style/StyleMap element in the KML document.

- Once published, such URLs must remain active forever - or at least as long as anyone might reference the particular style(s).

In general, it would be optimal, 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. There are, however, a few open issues, which are items for future work:

- KML features references styles, OGC Portrayal Service requests reference SLD (containing a set of layers with one or more styles). I.e., these are different resources that would need to be managed consistently in the registry.

- The current KML encoding rule does not support different styles per feature type. A similar filtering mechanism as in SLD/SE would need to be supported by the encoding rule to achieve the styling of features on the same level of granularity.

5.4 Implementation

5.4.1 ShapeChange

ShapeChange is a framework for processing UML models containing application schemas according to ISO 19109. Initially the main output of ShapeChange were GML application schemas, automatically derived from the UML model. While this is still a key output of ShapeChange, additional outputs have been added over time including feature catalogues, code list dictionaries, constraint representations, etc.

ShapeChange is implemented in Java and the core framework is available under the Gnu Public License.

5.4.2 Implementation in ShapeChange

A new target output class has been implemented in ShapeChange that implements the KML encoding rule so that an XSLT stylesheet is derived from the application schema that - applied to GML data of the same application schema - generates a KML document.

A shortened sample of the XSLT stylesheet generated by ShapeChange for the BuildingGeopoint feature type of the NAS OWS-7 profile used in the example in Annex A:

```
<stylesheet
    xmlns="http://www.w3.org/1999/XSL/Transform"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:kml="http://www.opengis.net/kml/2.2"
    xmlns:tds="http://metadata.dod.mil/mdr/ns/GSIP/2.0/tds/2.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<output indent="yes" method="xml"/>
<template match="/">
  <kml xmlns="http://www.opengis.net/kml/2.2">
    <Document>
      <open>0</open>
      <if xmlns="http://www.w3.org/1999/XSL/Transform"
          test="//tds:BuildingGeosurface">
        <Schema xmlns="http://www.opengis.net/kml/2.2"
             id="BuildingGeosurfaceSchema">
          <SimpleField name="type" type="string">
            <displayName>
                <![CDATA[
                <a href='https://nsgreg.nga.mil/as/view?i=100083'
             title='Building Geospatial Surface: A free-standing self-
             supporting construction that is roofed, usually walled, and is
             intended for human occupancy (for example: a place of work or
             recreation) and/or habitation. [desc] For example, a dormitory, a
             bank, and a restaurant.'><big><b><i>Building Geospatial
             Surface</i></b></big></a>
                ]]>
             </displayName>
          </SimpleField>
          <SimpleField name="conditionOfFacility" type="string">
            <displayName>
                <![CDATA[
                <a href='https://nsgreg.nga.mil/as/view?i=100083'
             title='Condition of Facility: The state of planning,
             construction, repair, and/or maintenance of the structures and/or
             equipment comprising a facility and/or located at a site, as a
             whole.'><b><i>Condition of Facility</i></b></a>
                ]]>
             </displayName>
          </SimpleField>
          ...
           <SimpleField name="uniqueEntityIdentifier" type="string">
            <displayName>
                <![CDATA[
                <a href='https://nsgreg.nga.mil/as/view?i=100083'
             title='Unique Entity Identifier: The globally unique and
             persistent identifier of an entity (for example: feature or
             event) instance as specified by a Uniform Resource Name (URN) in
             accordance with the Internet Engineering Task Force (IETF)
             RFC2396 and RFC2141. [desc] It is based on the Uniform Resource
             Identifier (URI), a compact string of characters for identifying
             an abstract or physical resource. The term Uniform Resource Name
             (URN) refers to the subset of URI that are required to remain
             globally unique and persistent even when the resource ceases to
             exist or becomes unavailable. The URN is drawn from one of a set
             of defined namespaces, each of which has its own set name
             structure and assignment procedures.'><b><i>Unique Entity
             Identifier</i></b></a>
                ]]>
             </displayName>
          </SimpleField>
        </Schema>
      </if>
      ...
      <!--Call styling template for each feature type-->
      <for-each xmlns="http://www.w3.org/1999/XSL/Transform"
         select="//tds:BuildingGeosurface">
        <call-template name="BuildingGeosurface"/>
      </for-each>
```

```
        ...
      </Document>
    </kml>
  </template>
  <!--Styling templates for each feature type-->
  <template name="BuildingGeosurface">
    <Placemark xmlns="http://www.opengis.net/kml/2.2">
      <attribute xmlns="http://www.w3.org/1999/XSL/Transform" name="id">
        <value-of select="@gml:id"/>
      </attribute>
      <choose xmlns="http://www.w3.org/1999/XSL/Transform">
        <when test="tds:geoNameCollection.memberGeoName.fullName">
          <name xmlns="http://www.opengis.net/kml/2.2">
            <value-of xmlns="http://www.w3.org/1999/XSL/Transform"
               select="./tds:geoNameCollection.memberGeoName.fullName[1]"/>
          </name>
        </when>
        <when test="gml:name">
          <name xmlns="http://www.opengis.net/kml/2.2">
            <value-of xmlns="http://www.w3.org/1999/XSL/Transform"
               select="./gml:name[1]"/>
          </name>
        </when>
        <otherwise>
          <name xmlns="http://www.opengis.net/kml/2.2">Building Geospatial
Surface</name>
        </otherwise>
      </choose>
      <visibility>1</visibility>
      <if xmlns="http://www.w3.org/1999/XSL/Transform"
        test="tds:geointAssuranceMetadata.currencyDateTime">
        <TimeStamp xmlns="http://www.opengis.net/kml/2.2">
          <when>
            <value-of xmlns="http://www.w3.org/1999/XSL/Transform"
               select="./tds:geointAssuranceMetadata.currencyDateTime[1]"/>
          </when>
        </TimeStamp>
      </if>
      <styleUrl>http://portele.de/styles.kml#s1</styleUrl>
      <ExtendedData>
        <SchemaData schemaUrl="#BuildingGeosurfaceSchema">
          <SimpleData name="type">
            <choose xmlns="http://www.w3.org/1999/XSL/Transform">
              <when test="gml:description">
                <value-of select="./gml:description"/>
              </when>
              <otherwise>A free-standing self-supporting construction that is
roofed, usually walled, and is intended for human occupancy (for example: a
place of work or recreation) and/or habitation. [desc] For example, a
dormitory, a bank, and a restaurant.</otherwise>
            </choose>
          </SimpleData>
          <if xmlns="http://www.w3.org/1999/XSL/Transform"
           test="(count(tds:conditionOfFacility) -
           count(tds:conditionOfFacility[@xsi:nil='true']))&gt;0">
            <SimpleData xmlns="http://www.opengis.net/kml/2.2"
               name="conditionOfFacility">
              <for-each xmlns="http://www.w3.org/1999/XSL/Transform"
               select="tds:conditionOfFacility">
                <if test="position()&gt;1"><![CDATA[<hr/>]]></if>
                <value-of select="."/>
              </for-each>
            </SimpleData>
          </if>
          ...
          <if xmlns="http://www.w3.org/1999/XSL/Transform"
           test="(count(tds:uniqueEntityIdentifier) -
           count(tds:uniqueEntityIdentifier[@xsi:nil='true']))&gt;0">
            <SimpleData xmlns="http://www.opengis.net/kml/2.2"
               name="uniqueEntityIdentifier">
              <for-each xmlns="http://www.w3.org/1999/XSL/Transform"
               select="tds:uniqueEntityIdentifier">
                <if test="position()&gt;1"><![CDATA[<hr/>]]></if>
                <value-of select="."/>
              </for-each>
            </SimpleData>
          </if>
        </SchemaData>
      </ExtendedData>
      <apply-templates xmlns="http://www.w3.org/1999/XSL/Transform"
        select="*/gml:Polygon|*/gml:LineString|*/gml:Point"/>
    </Placemark>
  </template>
  ...
</stylesheet>
```

5.5 Accessing KML data from a Web Feature Service

5.5.1 Introduction

The Web Feature Service interface supports a simple mechanism to access another output format than the default GML encoding. Using a different MIME type advertised in the service metadata in the outputFormat parameter returns the selected features not in GML, but in the requested format. For KML output, the applicable MIME type is

application/vnd.google-earth.kmz

which returns the KML data in a compressed archive.

Example http://services.interactive-instruments.de/ows7-kml/cgi-bin/ows7dev-kml?
version=1.1.0&
service=WFS&
request=getfeature&
typename=tds:BuildingGeopoint,tds:BuildingGeosurface,tds:CampGeosurface&
outputformat=application/vnd.google-earth.kmz&
bbox=18.53,-72.32,18.57,-72.28

5.5.2 Accessing large datasets

For small datasets, links like the one shown above may be added directly in Google Earth. However, for datasets with a larger area and a large number of features this quickly becomes a problem as Google Earth will access a large volume of data and not provide the performance / user experience expected by the user.

The natural mechanism provided by KML to address such issues are KML regions. That is, the area of the dataset is tesselated into smaller regions which are only loaded, if the region is visible in Google Earth in a certain size range. I.e., instead of registering a GetFeature request URL as in the example above, a KML document is provided that contains the regions, each with a KML network link to a WFS GetFeature request for the features in the region.

This approach also introduces potential performance issues (or requires a high-performance WFS). If the regions are small then often many will be visible at the same time, resulting in potentially a large number of parallel requests to the WFS. On the other hand, if the regions are large, a single request may take too long and degrade the user experience.

5.5.3 Representation in the "Places" tree in the Google Earth client

The Topographic Data Store (TDS) application schema consists of a large number of feature types (>100). Typically, a user may simply want to select a subset of all feature types to be visible. Considering that the TDS contains a two-level package hierarchy of the features types, the natural solution would be mapping the UML packages to KML folders. A shorted example of the resulting KML structure:

```
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>NAS Profile OWS-7 Haiti Geodatabase Schema</name>
    <visibility>1</visibility>
    <open>1</open>
    <description>This geodatabase schema defines the logical content for
Topographic Feature Data in the U.S. National System for Geospatial-
Intelligence (NSG). [desc] It specifically addresses selected ISO standards for
modeling features and surfaces (coverages), drawing on relevant military
standards, specifications and profiles established by the Defence Geospatial
Information Working Group (DGIWG).</description>
    <Snippet maxLines="0"/>
    <Folder>
      <name>Cultural</name>
      <visibility>1</visibility>
      <open>0</open>
      <description>Information about features on the landscape that have been
constructed by man. [desc] Excepted are the view groups 'Transportation',
'Ports and Harbours', 'Population', and their related features.</description>
      <Snippet maxLines="0"/>
      <Folder>
        <name>Power Generation and Transmission Facilities</name>
        <visibility>1</visibility>
        <open>0</open>
        <description>The buildings, non-building structures, and equipment
necessary for the production and distribution of electric power. [desc] The
power plant converts fuel to mechanical energy, which drives generators and
produces electricity. The main methods of generation are thermal energy (using
steam driven turbines), hydroelectric (using water pressure to drive turbo
generators) and internal combustion (direct conversion of mechanical to
electrical energy). The power distribution grid starts at the power plant and
ends with the subscriber. Networks of power transmission lines, carried on
transmission line pylons, transmit the electricity from the power plants
through the substations and transformer yards to the customer.</description>
        <Snippet maxLines="0"/>
        <NetworkLink>
          <name>Tiles</name>
          <visibility>1</visibility>
          <open>0</open>
          <refreshVisibility>0</refreshVisibility>
          <flyToView>0</flyToView>
          <Region>
            <LatLonAltBox>
              <north>20.0</north>
              <south>17.0</south>
              <east>-65.0</east>
              <west>-75.0</west>
            </LatLonAltBox>
            <Lod>
              <minLodPixels>512</minLodPixels>
              <maxLodPixels>-1</maxLodPixels>
            </Lod>
          </Region>
          <Link>
            <href>http://services.interactive-instruments.de/ows7-
kml/kml2/wfskml.php</href>
            <viewRefreshMode>onRegion</viewRefreshMode>

<viewFormat>BBOX=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth]&amp;LOOKAT=[look
atLon],[lookatLat],[lookatRange],[lookatTilt],[lookatHeading]&amp;CAMERA=[camer
aLon],[cameraLat],[cameraAlt]&amp;LOOKATTERR=[lookatTerrainLon],[lookatTerrainL
at],[lookatTerrainAlt]&amp;VIEW=[horizFov],[vertFov],[horizPixels],[vertPixels]
,[terrainEnabled]</viewFormat>

<httpQuery>CLIENTINFO=[clientVersion];[kmlVersion];[clientName];[language]&amp;
TYPENAME=tds:PowerStationGeosurface,tds:PowerStationGeopoint</httpQuery>
          </Link>
        </NetworkLink>
      </Folder>
      <Folder>
        <name>General Structures</name>
        <visibility>1</visibility>
        <open>0</open>
        <description>Buildings and their components, non-building structures
and man-made barriers (for example: walls and fences). [desc] They are
geographically widespread and may be located in rural, urban and/or industrial
settings.</description>
        <Snippet maxLines="0"/>
        <NetworkLink>
          <name>Tiles</name>
          <visibility>1</visibility>
          <open>0</open>
          <refreshVisibility>0</refreshVisibility>
          <flyToView>0</flyToView>
          <Region>
            <LatLonAltBox>
              <north>20.0</north>
              <south>17.0</south>
              <east>-65.0</east>
              <west>-75.0</west>
            </LatLonAltBox>
            <Lod>
              <minLodPixels>512</minLodPixels>
              <maxLodPixels>-1</maxLodPixels>
            </Lod>
          </Region>
          <Link>
            <href>http://services.interactive-instruments.de/ows7-
kml/kml2/wfskml.php</href>
            <viewRefreshMode>onRegion</viewRefreshMode>

<viewFormat>BBOX=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth]&amp;LOOKAT=[look
atLon],[lookatLat],[lookatRange],[lookatTilt],[lookatHeading]&amp;CAMERA=[camer
aLon],[cameraLat],[cameraAlt]&amp;LOOKATTERR=[lookatTerrainLon],[lookatTerrainL
at],[lookatTerrainAlt]&amp;VIEW=[horizFov],[vertFov],[horizPixels],[vertPixels]
,[terrainEnabled]</viewFormat>

<httpQuery>CLIENTINFO=[clientVersion];[kmlVersion];[clientName];[language]&amp;
TYPENAME=tds:BuildingGeopoint,tds:BuildingGeosurface</httpQuery>
          </Link>
        </NetworkLink>
      </Folder>
      <Folder>
        <name>Burial Sites</name>
        <visibility>1</visibility>
        <open>0</open>
        <description>A structure within which a corpse is entombed or an area
of ground in which the dead are buried. Also includes the buildings and
services used in the preparation and disposal of the dead and related
activities. [desc] It may be either a site where remains of ancient
civilizations have been discovered or a modern day facility.</description>
        <Snippet maxLines="0"/>
        <NetworkLink>
          <name>Tiles</name>
          <visibility>1</visibility>
          <open>0</open>
          <refreshVisibility>0</refreshVisibility>
          <flyToView>0</flyToView>
          <Region>
            <LatLonAltBox>
              <north>20.0</north>
              <south>17.0</south>
              <east>-65.0</east>
              <west>-75.0</west>
            </LatLonAltBox>
            <Lod>
              <minLodPixels>512</minLodPixels>
              <maxLodPixels>-1</maxLodPixels>
            </Lod>
          </Region>
          <Link>
            <href>http://services.interactive-instruments.de/ows7-
kml/kml2/wfskml.php</href>
            <viewRefreshMode>onRegion</viewRefreshMode>

<viewFormat>BBOX=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth]&amp;LOOKAT=[look
atLon],[lookatLat],[lookatRange],[lookatTilt],[lookatHeading]&amp;CAMERA=[camer
aLon],[cameraLat],[cameraAlt]&amp;LOOKATTERR=[lookatTerrainLon],[lookatTerrainL
at],[lookatTerrainAlt]&amp;VIEW=[horizFov],[vertFov],[horizPixels],[vertPixels]
,[terrainEnabled]</viewFormat>

<httpQuery>CLIENTINFO=[clientVersion];[kmlVersion];[clientName];[language]&amp;
TYPENAME=tds:CemeteryGeopoint,tds:CemeteryGeosurface</httpQuery>
          </Link>
        </NetworkLink>
      </Folder>
    </Folder>
    ...
</Document>
</kml>
```

However, the results have not been satisfactory. With about 30 leaf packages this results in many parallel connections to the WFS per region. Considering that in a single view several regions may be visible this quickly turns into several hundreds of parallel WFS requests for a single client. As a result this approach does not scale very well – or at least it requires a high-performance server architecture.

There are several measures that could be made to considered to make this work in an operational context, too:

- Higher aggregation. Instead of using the leaf packages, maybe just use the first package layer to reduce the number of KML folders (ie. parallel requests)
- Improved implementation. For the testing purposes we converted the encoding rule into XSLT scripts that convert a GML file from the WFS to its associated KML file. Since the model is quite big this results in a large XSLT script (upto 2MB) which is invoked in every request. While further improvements to the XSLT execution are possible, for a real fit-for-purpose implementation the KML should be encoded directly by the WFS instead of the "detour" via GML.
- For data that does not change too often, the KML for a tile/region could be stored as a KMZ file so that only a file access is necessary. These KMZ files could be derived directly from the WFS as a cache and placed in the file system by a periodic process, if needed.

6 Constraints expressed using Schematron

6.1 Overview

OWS-5 and OWS-6 have started to investigate the use of Schematron to represent OCL constraints from the application schema to enable the validation of GML data beyond the syntactic validation using XML Schema. It was found that the general approach was feasible and useful. Building upon these results, OWS-7 aimed at a more comprehensive approach. The following items were addressed:

6.2 Comprehensive conversion rules OCL to Schematron

In contrast to the implementation in OWS-5 and OWS-6, which was based on the recognition of a selected set of expression patterns, the new “OCL to Schematron” compilation has been based on a general translation process. Basic OCL 2.2 language constructs are transformed into equivalent schema constructs. Nevertheless, there are still some limitations as

- the new ShapeChange OCL does not comprise the full language definition, and
- not everything, which can be written down in OCL, can be also expressed in XPath 1.0 and Schematron.

The conversion rules and the XPath1.0-caused limitations of the translation process are documented in Annex B.

6.3 Implementation in ShapeChange

The existing XML Schema target output class that is implemented in ShapeChange has been enhanced with the OCL to Schematron conversion rules. This makes use of an OCL 2.2 parser that recently has been implemented in ShapeChange. The XML Schema target class has been supplemented with additional code to perform the Schematron code generation on the basis of an OCL syntax tree generated by the new OCL 2.2 parser.

6.4 Validation using the NAS model

The OCL-to-Schematron translation process has been extensively tested on the NAS model. Following the necessary adaption of the already existing OCL constraints to the OCL 2.2 standard, many new OCL constraints have been added. These new OCL constraints cover nearly all of the cases in Annex B.

New requirements have been detected during the validation process and have been implemented accordingly. The generated output documents, including error messages and Schematron code have been carefully reviewed and discussed by the participants.

However, lacking corresponding GML instance documents of NAS data, neither the XML Schema representation nor the Schematron code has been tested against data instances.

6.5 Extensions implemented during the validation process

The validation process brought forward a couple of additional functional requirements. Most of these could be treated by extending the implementation of the OCL-to-Schematron translation in ShapeChange.

6.5.1 Nested error messages

The analysis of OCL invariants by the new OCL parser of ShapeChange frequently generates more than one error message for one OCL invariant.

In the pre-OWS-7 version of ShapeChange this had been handled by first emitting an introductory message identifying the constraint and the class context, followed by one or more messages explaining the exact nature of the problems detected.

Example:

```
<Error>One or more errors encountered in OCL constraint in class Installation : LengthValMeta ...</Error>
<Error>Line/column(s) 6/362: Unrecognized syntax [!] encountered and ignored.</Error>
<Error>Line/column(s) 6/483: Identifier, literal or bracketed expression expected preceding operator &quot;=&quot;
token. Invalid assumed.</Error>
<Error>Line/column(s) 6/485-502: Closing bracket ')' expected preceding identifier
&quot;VoidNumValueReason&quot; token, assumed.</Error>
<Error>Line/column(s) 6/485-502: Extra tokens following OCL expression ignored, starting with 'identifier
&quot;VoidNumValueReason&quot;'.</Error>
```

In analyzing the error reports from the OCL attached to NAS classes this format was deemed not sufficiently comprehendible.

It was changed to a nested format as follows:

```
<Error message="One or more errors encountered in OCL constraint in class Installation : LengthValMeta ...">
  <Detail message="Line/column(s) 6/362: Unrecognized syntax [!] encountered and ignored."/>
  <Detail message="Line/column(s) 6/483: Identifier, literal or bracketed expression expected preceding operator
&quot;=&quot; token. Invalid assumed."/>
  <Detail message="Line/column(s) 6/485-502: Closing bracket ')' expected preceding identifier
&quot;VoidNumValueReason&quot; token, assumed."/>
  <Detail message="Line/column(s) 6/485-502: Extra tokens following OCL expression ignored, starting with
'identifier &quot;VoidNumValueReason&quot;'."/>
</Error>
```

6.5.2 Identity comparisons between items of non-basic type

As it turned out it was necessary to provide support for equality comparisons on object-valued items. This had initially been ruled out, because it is hard to compile to XPath 1.0.

The following invariant expresses that the current feature of type *SoundingMetadata* requires that there is an associated *HydroVertDimMeta* object.

This would usually have been expressed by an invariant such as

inv SoundingMetadata_ValidUse: hydroVertDimMeta->notEmpty()

where *hydroVertDimMeta* points to the associated *HydroVertDimMeta* object.

However, this *hydroVertDimMeta* role had been marked non-navigable in the model, which disallows its use in the constraint. So this had to be re-formulated "from the other side" using the *allInstances()* operation.

inv SoundingMetadata_ValidUse:
  HydroVertDimMeta.allInstances().soundingMetadata->exists(sm|sm=self)

To do the translation of the sub-expression *sm=self*, equality comparisons had to be implemented also for objects. The implementation was added for comparison arguments, which have identity (such as FeatureTypes) and results in the use of the XPath function

generate-id(node)

on the arguments. The function derives a unique string identifier from an element.

Note that this is an XSLT-extension to XPath and not a native XPath 1.0 one. This might lead to problems on Schematron implementations, which are not based on XSLT.

Comparisons between class instances, which do not carry identity (such as DataTypes), are still refused in the Schematron code generation.

6.5.3 Inheritance of constraints

The implementation provided no support for inheriting constraints attached to the classes in the model, because it was not clear whether inheritance was indeed needed and how constraints were supposed to override.

As it turned out, inheritance of constraints was indeed a requirement.

The implementation permits to override constraints by their name. The name of an OCL invariant is specified following the *inv* reserved word.

Example:

> *inv SoundingMetadata_ValidUse:*
> *HydroVertDimMeta.allInstances().soundingMetadata->exists(sm|sm=self)*

The name of this invariant constraint is "SoundingMetadata_ValidUse".

6.5.4 Support of types derived from ISO 19103 basic types

There is currently no specification contained in ISO 19103, which would put into order how the basic types defined in there (such as CharacterString) relate to the built-in OCL data types (such as String). Lacking proper definitions, the new OCL parser makes reasonable assumptions about a correspondence between ISO 19103 types and those built-in in OCL. ISO 19103 types are generally mapped to built-in types in a non-surprising way.

This allows to write a comparison such as *someProperty='xxx'* for a CharacterString *someProperty*. Without that mapping such a comparison would result in the constant value *false*.

As it turned out, the implemented mechanism was too weak. The NAS model specializes the basic ISO 19103 types and uses the derived types in its data declarations, which had previously not been foreseen.

So, general support for types derived from the basic types of ISO 19103 had to be added.

6.5.5 Mapping of enumeration and codelist values

There is a mismatch between the modeling of "reasons" for missing data in the NAS model and in GML 3.2.1.

The different reason vocabularies map as follows:

GSIP Reason	Integer	gml:NilReasonType	GML 3.2.1 definition
noInformation	-999999	missing	the correct value is not readily available to the sender of this data. Furthermore, a correct value may not exist
valueSpecified	995		
notApplicable	998	inapplicable	there is no value

GSIP Reason	Integer	gml:NilReasonType	GML 3.2.1 definition
other	999	other:other	other brief explanation, where text is a string of two or more characters with no included spaces

The first column describes the values used in the model, the third column describes the values as expected in GML documents representing the data. OCL constraints would refer to the values in the first column, while the generated Schematron code need to use the values in the third column.

So, some translation mechanism is required, which allows to map the values, while performing the OCL to Schematron translation.

In GML the type definition is

```
<simpleType name="NilReasonType">
  <union memberTypes="gml:NilReasonEnumeration anyURI"/>
</simpleType>
```

where *gml:NilReasonEnumeration* contains the values *missing, inapplicable, other:other*.

Concerning the *anyURI* part of the union GML 3.2.1 is states:

> “anyURI which should refer to a resource which describes the reason for the exception”

This opens up the possibility to choose to assign more detailed semantics to the standard values provided. The URI method enables a specific or more complete explanation for the absence of a value to be provided elsewhere and indicated by-reference in an instance document.

It has been decided to go this way. These are the URIs employed:

noInformation:	http://metadata.dod.mil/mdr/ns/GSIP/codelist/VoidValueReason/noInformation
valueSpecified:	http://metadata.dod.mil/mdr/ns/GSIP/codelist/VoidValueReason/valueSpecified
notApplicable:	http://metadata.dod.mil/mdr/ns/GSIP/codelist/VoidValueReason/notApplicable
other:	http://metadata.dod.mil/mdr/ns/GSIP/codelist/VoidValueReason/other

The implementation has been carried out by providing another tagged value named

resourceURI

for that purpose. If applied to the values of enumerations or codelists, the Schematron code generator simply uses the values of those tagged values in place of the original values if the former are present and not empty.

6.6 Issues found in need of explanation during validation

6.6.1 Uses of the size operator

Let *placeName* be a String.

A reference to its length (limiting it to 20 characters) is expressed as follows:

placeName.size() <= 20

To refer to the cardinality of *placeName* and compare it to the constant 3, you write:

placeName->size() = 3

This would test, whether *placeName* refers to exactly three object instances.

6.6.2 The use of isEmpty, notEmpty

Checking a if a property is set or not set or not is done with *isEmpty* or *notEmpty*, respectively, regardless whether the *property* has a cardinality ≤ 1 or > 1.

property->isEmpty()

evaluates to *true* if *property* contains no value and is therefore equivalent to

property->size()=0

Though OCL treats objects with a cardinality ≤ 1 as single objects, it automatically promotes these single objects to a collection, whenever the -> operator is applied.

6.6.3 Testing collections of objects

While objects with a cardinality ≤ 1 can be directly used in comparisons or with other operators as in

inv: identification.entityIdentifierType<>EntityIdentifierType::VerticalObstIdentifier

Objects with a cardinality > 1 always need an iterator such as *exists* or *forAll* being employed, depending on the intended quantification semantics.

If *identification* in the example above were of cardinality > 1, and you need that all instances of it are different from the given codelist value, then the following formulation would be effective:

```
inv: identification->forAll( x |
   x.entityIdentifierType <> EntityIdentifierType::VerticalObstIdentifier )
```

Of course, the latter construct can also be applied if the cardinality is ≤ 1.

6.6.4 The use of isUnique

isUnique is a special iterator. As with all iterators its left-hand operand (its self) is a collection. An expression on the members of that collection is tested for uniqueness, which means that the values of that expression must not repeat.

The implementation of isUnique in the OCL to Schematron translator is rather restricted and only allows property references as the possible syntax for expressions. See B.1 for a discussion of these issues.

General format:

x->isUnique(y|expr(y))

In the case of basic-type values with a cardinality > 1, which are being tested for uniqueness, the expression in the iterator body typically is identity.

For example, in

inv: featureFunction.valuesOrReason.values->isUnique(x|x)

the property *featureFunction* has a property *valuesOrReason*, which is a Union, one property of which is *values*, which has cardinality > 1 and is of a basic type.

The phrase *isUnique(x|x)* needs to be written as shown (of course with any other variable identifier in place of x). It is not possible to abbreviate this to *isUnique()*.

It is also not possible to rephrase the example above to

inv: featureFunction.valuesOrReason->isUnique(x|x.values)

This expresses wrong semantics, because the "collection" *valuesOrReason* only has one member, which means that the concept of uniqueness does not make any sense. Moreover, this would also be rejected, because there is a restriction in the translation process. This is because the expression *x.values* leads to a collection, and no sensible way to compare collection valued items had been found, which could be expressed in XPath 1.0.

6.6.5 The effects of navigabilty

Properties, which designate a non-navigable role of an association, cannot be used in OCL constraints.

ISO 19103 D.7.2: "If this is important to the model, the association should be two-way navigable to make enforcement of the constraint more tenable. In other words, a one-way relation implies a certain "don't care" attitude towards the non-navigable end."

It is usually possible to express the required invariant "from the other side" (from the instances of the referenced objects) by using the *allInstances* function. See 6.5.2 for an example.

6.6.6 Equivalence and antivalence

Equivalence and antivalence are usually hard to express using the standard logic particles *and*, *or* and *not*. These expressions usually require that the operands need to be written more than once, which is unfortunate, if the operands themselves are expressions.

For example equivalence of the logical items *a* and *b* can be expressed by:

(a and b) or (not a and not b)

and antivalence by:

(a and not b) or (not a and b)

Fortunately, OCL also supports the logical *xor* particle, which directly corresponds to antivalence. So, antivalence can be directly written as:

a xor b

This allows denoting equivalence as:

not a xor b ≡ a xor not b ≡ not (a xor b)

Equivalence, however, can also be directly expressed in OCL by using the equality operator = between Boolean expressions. Thus, equivalence can also be written as:

a = b

Of course, antivalence can also be expressed by the <> operator.

6.6.7 String matching and REGEX language

The implemented *matches* function is an extension to OCL 2.2. See B.2 for further details.

Signature:

matches(pattern:String) : Boolean

By means of the ShapeChange configuration matches can be either mapped to Java *java.util.regex.Pattern.matches* or to the XPath 2.0 function *fn:matches*.

Of course, the specific details of the REGEX language to be used in the pattern are as in the environment which is to be used. The individual differences between the REGEX languages realized by Java and XPath 2.0 have not been investigated in depth, and have not been compared to the REGEX variant of XML Schema's pattern facet.

It can be expected that all three REGEX languages are quite similar.

6.6.8 The necessity of guard preconditions

It is usually necessary to account for the multiplicity of properties when formulating OCL constraints.

This is usually enforced by the syntax and semantics of OCL, whenever properties display a cardinality > 1 – you need to apply iterators such as *exists* or *forAll* or similar constructs, depending on the quantification semantics you have in mind.

However, the need to honor the multiplicity of properties is also required for the case of cardinality ≤ 1, where OCL permits to address the property without any surrounding iterator construct.

Let *width* own a cardinality ≤ 1. If you have an OCL invariant such as

> *inv: width < 17*

and the *self* instance carries an empty *width* instance, then in OCL *width* assumes the value *null*, which is an instance of *OclVoid*. Comparing *null* with the "<" operator (which is actually a *width.<(17)* operation call) will result in the value *invalid*, which is an instance of *OclInvalid*.

The result of the expression is therefore *invalid* (of type *OclInvalid*), which is clearly different from *true* (of type Boolean), so the invariant will fail. However, this is most probably not the intended semantics.

Therefore, you will need to guard the intended invariant against the absence of *width* with an *implies* particle:

> *inv: width->notEmpty() implies width < 17*

In this formulation you will get a *true* result, if *width* has no value. The check against 17 is only effective if a value is indeed present.

The guard can only be left away, if the cardinality of *width* is known to be exactly = 1.

Please note that the formulation

> *inv: width->forAll(x | x < 17)*

which would have been used for cardinality > 1, can also be used in the case of cardinality ≤ 1. It has the same effect as the formulation using *implies* above. The invariant is fulfilled if *width* has no value.

6.7 Open issues

6.7.1 Full treatment of xlink:href

The OCL invariant

inv: self.runway->notEmpty()

is currently translated into

//[concat('#',@gml:id)=current()/agoas:runway/@xlink:href*

This picks those elements from the GML instance document, each of which carries a *gml:id* attribute, the value of which, when prefixed by # happens to be identical to the *xlink:href* attribute on at least one of the property *agoas:runway* instances on the current feature. If the nodeset of elements picked this way is not empty, then the XPath expression is finally converted to the Boolean *true*, which means that the assertion holds.

This translation reflects the semantics of OCL. The expression *self.runway* in OCL means the set of objects referenced by the *runway* property and this is just, what the corresponding *//*[...]* expression in XPath finds.

The translation is under the assumption that *xlink:href* references prefix the id with a #. The implementation permits to configure other prefixes and postfixes if required.

The open problem is that the GML document to be checked also requires the referenced objects to be contained.

There are two possible solutions to deal with this:

1. For the *notEmpty()* operator less than the above could be checked. We could for example check whether the property is there, or perhaps the property together with @xlink:href is there. In this case we would simply translate to

   agoas:runway/@xlink:href

   However, this is not a general solution, because it must be restricted to special operators. *notEmpty(), isEmpty(), size()* might work this way, because they need nothing but the number of items. Definitely *self.runway->select(t|someexpression(t))* or any use of the *exists()* or *forAll()* iterators does not work, because it requires access to the contents of the objects.

2. It might also be possible to extend the test into external documents.

   For this the XPath 1.0 function *document()* might be employed, which, if supplied with a set of URLs (of XML documents) delivers a nodeset of root nodes of the documents addressed by these URLs.

   The problem to solve here is how to extract the URL of an referenced GML instance document (presumably created on the fly by some service invocation) from an @xlink:href value. Supposed this could be done by some magic *hrefToURL()* function, then the notation would be straightforward as follows:

   document(hrefToURL(agoas:runway/@xlink))
   //[concat('#',@gml:id)=current()/agoas:runway/@xlink:href*

Though this will probably work in principle, its usefulness in practice still needs to be proven, because the performance of such a construct might be quite low.

6.7.2 Implementation of let expressions

The OCL construct

let variable : type = expression, ... in expression-using-variable

permits to define and initialize variables and subsequently make use of these in an embedded expression. The idea is a short-hand notation for expressions, which contain more than one occurrences of a sub-expression.

Translating *let* expressions in the OCL-to-Schematron context would be possible in principal, but has not been done, because it is not a necessary construct. Everything, which can possibly be expressed using a *let*, can also be expressed without.

As it turned out, however, the use of OCL constraints in the NAS model produced quite lengthy constraints, the readability of which would benefit a lot from providing *let*.

A general translation of let to Schematron and XPath 1.0 can be done:

- *let* expressions at the outmost level of expressions are relatively simple to translate, because Schematron syntax itself provides a let construct.
- *let* expressions in deeper nesting levels, where the variable initialization or the expression in the body refers to bound variables of iterators, require substitution of the translated expression in all places, where the variable is used.

With XPath 2.0 instead of XPath 1.0 a complete translation is possible.

6.7.3 Consideration of enclosed ISO 19139 encodings

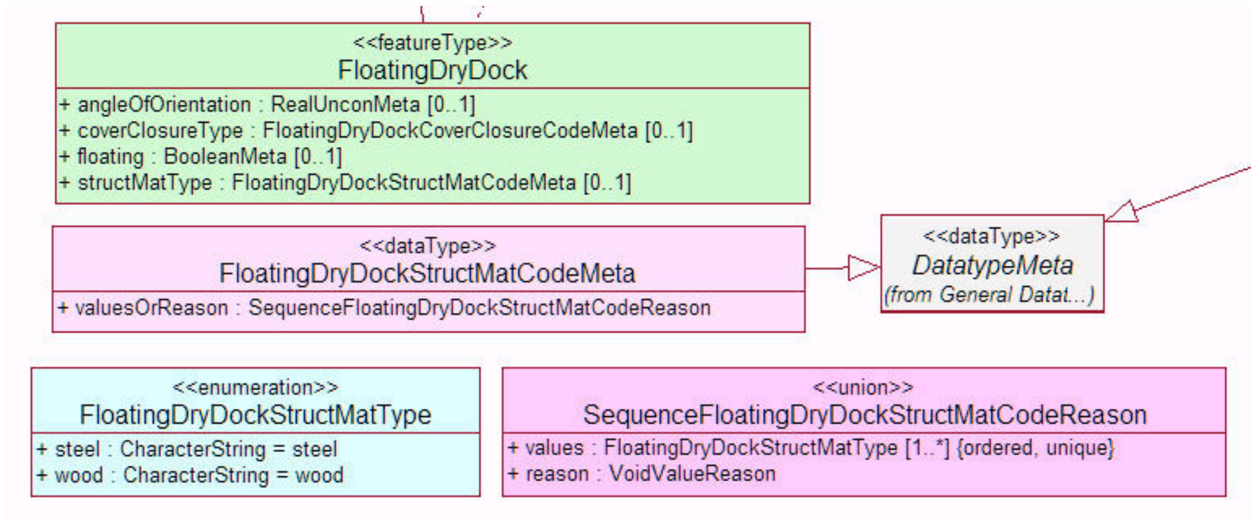
Currently the Schematron code generation supports only the 19136 encoding rule where simple values are a text node child of the property element.

The extra element level from the 19139 encoding rule and the 19139 code list type are not supported, so there is additional work required to make this work with metadata profiles.

6.7.4 The treatment of nillable attributes with cardinality > 1

This problem is not an OCL issue in the first place. It is mainly concerned with the UML-to-GML mapping of attributes, which are supposed to be both nillable and multiple.

Example:


<<featureType>>
FloatingDryDock
+ angleOfOrientation : RealUnconMeta [0..1]
+ coverClosureType : FloatingDryDockCoverClosureCodeMeta [0..1]
+ floating : BooleanMeta [0..1]
+ structMatType : FloatingDryDockStructMatCodeMeta [0..1]
<<dataType>>
FloatingDryDockStructMatCodeMeta
+ valuesOrReason : SequenceFloatingDryDockStructMatCodeReason
<<dataType>>
DatatypeMeta
(from General Datat...)
<<enumeration>>
FloatingDryDockStructMatType
+ steel : CharacterString = steel
+ wood : CharacterString = wood
<<union>>
SequenceFloatingDryDockStructMatCodeReason
+ values : FloatingDryDockStructMatType [1..*] {ordered, unique}
+ reason : VoidValueReason

In ShapeChange this is currently translated to XML Schema by attaching the multiplicity of the absorbed *values* attribute to *valuesOrReason*, which is made nillable and which gets an additional *nilReason* attribute.

This is implemented by the following XML Schema fragment:

```
<element name="valuesOrReason" nillable="true" maxOccurs="unbounded">
  <complexType>
    <simpleContent>
      <extension base="nas:FloatingDryDockStructMatTypeType">
        <attribute name="nilReason" type="gml:NilReasonType"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
```

This is according to the encoding rule extension that has been agreed in earlier OWS phases. It applies the normal mapping for the values property (just that it is mapped to the *valuesOrReason* property element which in addition is made nillable). Property values in the GML encoding rules are generally represented by elements.

Instance document are supposed to represent either a “normal” GML property, where multiplicity is applied by repeating the element, as in the following snippet:

```
<nas:structMatType>
  <nas:FloatingDryDockStructMatCodeMeta>
    <nas:valuesOrReason>steel</nas:valuesOrReason>
    <nas:valuesOrReason>wood</nas:valuesOrReason>
  </nas:FloatingDryDockStructMatCodeMeta>
</nas:structMatType>
```

Or alternatively, if the property is nil, one instance of the property with *xsi:nil* and *nilReason* applied:

```
<nas:structMatType>
  <nas:FloatingDryDockStructMatCodeMeta>
    <nas:valuesOrReason xsi:nil="true" nilReason="missing"/>
  </nas:FloatingDryDockStructMatCodeMeta>
</nas:structMatType>
```

The problem with this is that it is only an agreement; it is not enforced automatically by any constraint mechanism. XML Schema provided with the schema fragment above would allow mixed representations, such as:

```
<nas:structMatType>
  <nas:FloatingDryDockStructMatCodeMeta>
    <nas:valuesOrReason>steel</nas:valuesOrReason>
    <nas:valuesOrReason xsi:nil="true" nilReason="missing"/>
  </nas:FloatingDryDockStructMatCodeMeta>
</nas:structMatType>
```

There is no XML schema constraint, which would guarantee that only the agreed variants can be applied. However a Schematron constraint could enforce this:

context nas:FloatingDryDockStructMatCodeMeta
  inv: nas:valuesOrReason/@xsi:nil=true implies count(nas:valuesOrReason)=1

Such an invariant needs not be explicitly specified. It can be generated out of the information contained UML schema.

The point discussed and which eventually needs to be decided was whether the mapping agreed in earlier OWS phases and currently applied has been defined correctly.

The argument is:

Since the multiplicity is on the *values* attribute of the <<union>> (see UML diagram above), it should go directly into the type of *values*. This means that *nas:valuesOrReason* would receive a list type.

In this case the value and nil instance examples would look like this:

```
<nas:structMatType>
  <nas:FloatingDryDockStructMatCodeMeta>
    <nas:valuesOrReason>steel wood</nas:valuesOrReason>
  </nas:FloatingDryDockStructMatCodeMeta>
</nas:structMatType>

<nas:structMatType>
  <nas:FloatingDryDockStructMatCodeMeta>
    <nas:valuesOrReason xsi:nil="true" nilReason="missing"/>
  </nas:FloatingDryDockStructMatCodeMeta>
</nas:structMatType>
```

The good thing in this representation is that it would be syntactically enforced by XML Schema alone. Since the multiplicity is expressed in the type of *nas:valuesOrReason* the element itself needs to occur only once. There cannot be a mix of representations.

There are counterarguments, however:

- In the GML encoding rule multiplicity is expressed by *maxOccurs* on the element, which stands for the property, and
- if an exception to this rule would be made, it could only be made for simple data. According to GML encoding rules it is not an option to use multiple instances of complex data constructs in a property.
- OCL rules, which would refer to the list data type, could not be properly translated to XPath 1.0 and Schematron. See explanations below.

XPath 1.0 does not possess the functionality to process lists in the extent required for expressing the agreed subset of OCL. Though there are a few functions in XPath 1.0, which allow doing elementary operations such as split a string at a certain position, there is no vehicle to combine these operation into functions, which would realize OCL operations like unique(), exists(), etc. You can define and call parameterized templates in XSLT, but you cannot do this in XPath alone and by using the language items of Schematron.

We would need to implement each single OCL operation in Java extension code. Things are very different in XPath 2.0. XPath 2.0 (which is not the basis of standard Schematron) would offer all the functionality necessary to process lists and more.

6.7.5 The use of XPath 2.0 functionality

The current implementation of OCL-to-Schematron in ShapeChange targets the ISO Schematron standard (ISO/IEC 19757-3), which is based on XPath 1.0.

Though XPath 1.0 looks like a powerful language at the first glance, it does by far not allow expressing all OCL constructs, and those which it allows expressing sometimes require a bag of tricks – tricks which often lead to constraint code of low performance. See B.4.2 as an example of the latter.

XPath 2.0 is much more capable than XPath 1.0. None of the difficulties discussed in B.1 would occur, when we would have to translate to XPath 2.0 expressions. The true limits of XPath 2.0 still have to be investigated, but from the first view it appears as if OCL might be possible to translate to XPath 2.0 completely.

Though currently XPath 1.0 is the basis of ISO Schematron, there are already a few implementations (such as SAXON 9), which permit to use XPath 2.0 expressions in Schematron schemas.

Moreover, there is a revision of ISO Schematron underway that allows Schematron to be used with XPath 2.0. The revised version of the standard has already successfully gone through its first ballot at ISO. It can be expected to come out in early 2011.

Annex A
(normative)

KML encoding rule

A.1 General encoding requirements

A.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 A.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.

A.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).

A.1.3 Exchange metadata

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

A.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.

A.1.5 Update mechanism

The general KML mechanisms for updates apply.

A.2 Input data structure

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

A.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”.

A.4 Conversion rules

A.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. The document may be packaged with other documents (e.g., styles) into a KMZ document.

NOTE In order to support self-contained KMZ files it may be appropriate to support two different style URLs, one absolute and one relative within a KMZ file. To be decided after implementation experience.

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.
  - 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[1]

  The CDATA block contains the following value:

    - If the feature instance has a “gml:description” value, this value is used.
    - Alternatively, if the feature type has a tagged value “description”, its value is used.
    - 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.

[1] 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;”.

  - 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.

A.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>”.

A.5 Example <informative>

Example of a KML document based on a NAS-conformant application schema for topographic data and one building feature:

```
<kml xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:gml="http://www.opengis.net/gml/3.2"
     xmlns:kml="http://www.opengis.net/kml/2.2"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns="http://www.opengis.net/kml/2.2">
   <Document>
      <open>0</open>

      <Schema id="BuildingGeosurfaceSchema">
         <SimpleField name="type" type="string">
```

```
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Building Geospatial Surface: A free-standing self-supporting construction that is roofed, usually walled, and is intended for human occupancy (for example: a place of work or recreation) and/or habitation. [desc] For example, a dormitory, a bank, and a restaurant.'&gt;&lt;big&gt;&lt;b&gt;&lt;i&gt;Building Geospatial Surface&lt;/i&gt;&lt;/b&gt;&lt;/big&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="conditionOfFacility" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Condition of Facility: The state of planning, construction, repair, and/or maintenance of the structures and/or equipment comprising a facility and/or located at a site, as a whole.'&gt;&lt;b&gt;&lt;i&gt;Condition of Facility&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="controllingAuthority" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Controlling Authority: The controlling authority responsible for a facility or site. [desc] Controlling authorities may be distinguished by organizational level (for example: national, sub-national, or military district) and/or type (for example: private or public).'&gt;&lt;b&gt;&lt;i&gt;Controlling Authority&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="featureFunction-1" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Feature Function [1]: The purpose(s) of, or intended role(s) served by, the feature.'&gt;&lt;b&gt;&lt;i&gt;Feature Function [1]&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="featureFunction-2" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Feature Function [2]: The purpose(s) of, or intended role(s) served by, the feature.'&gt;&lt;b&gt;&lt;i&gt;Feature Function [2]&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="geointAssuranceMetadata.currencyDateTime" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Currency Date and Time: The date and, optionally, time assigned to a data set (for example: the digital representation of a single feature or a set of features) as a whole that provides an overall assessment of its currency. [desc] Often known as the as of date, the overall currency of a data set is affected by knowledge of the source(s) and processes used to define the location, geometry, and other properties (attributes and associations) of the digital representation.'&gt;&lt;b&gt;&lt;i&gt;Currency Date and Time&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="geointAssuranceMetadata.processStep.processStepDescription" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Process Step Description: A narrative or other textual description of a process step, including related processing parameters and/or tolerances. [desc] A process step is an event or transformation in the life of a dataset that is used to define, review and/or update the digital representation of a feature and/or attribute. No restriction is placed on the length of the description.'&gt;&lt;b&gt;&lt;i&gt;Process Step Description&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="geointAssuranceMetadata.processStep.processStepProcessor" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083' title='Process Step Processor: The identity of, and means of communication with, person(s) and/or organisation(s) associated with a process step. [desc] A process step is an event or transformation in the life of a dataset that is used to define, review and/or update the digital representation of a feature and/or
```

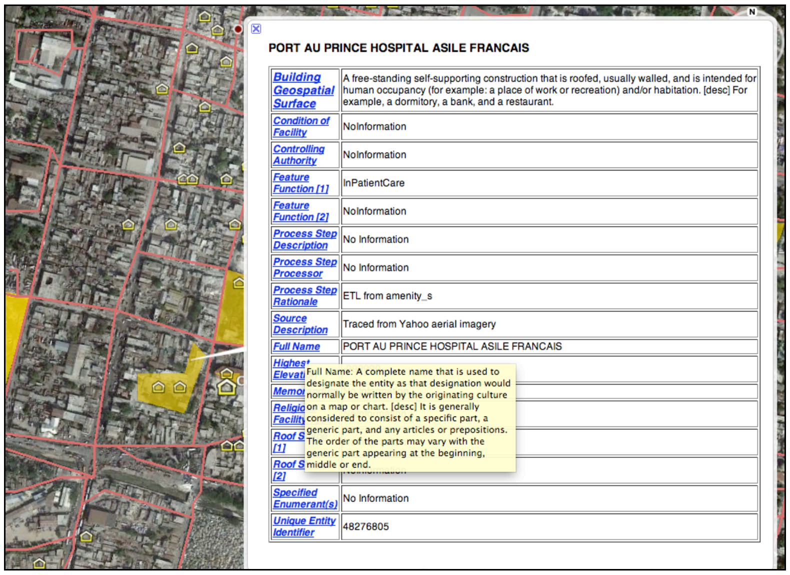
```
attribute.'&gt;&lt;b&gt;&lt;i&gt;Process                                    Step
Processor&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField
name="geointAssuranceMetadata.processStep.processStepRationale" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Process Step Rationale: A narrative or other textual description of the
requirement or purpose for a process step. [desc] A process step is an event or
transformation in the life of a dataset that is used to define, review and/or
update the digital representation of a feature and/or attribute. No restriction
is placed on the length of the rationale.'&gt;&lt;b&gt;&lt;i&gt;Process Step
Rationale&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField
name="geointAssuranceMetadata.processStep.source.sourceDescription"
                      type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Source Description: A description of the data set that was used to define
the digital representation of the feature or data set. [desc] No restriction is
placed   on   the   length   of   the   description.'&gt;&lt;b&gt;&lt;i&gt;Source
Description&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField           name="geoNameCollection.memberGeoName.fullName"
type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Full Name: A complete name that is used to designate the entity as that
designation would normally be written by the originating culture on a map or
chart. [desc] It is generally considered to consist of a specific part, a
generic part, and any articles or prepositions. The order of the parts may vary
with    the    generic    part    appearing    at    the    beginning,    middle    or
end.'&gt;&lt;b&gt;&lt;i&gt;Full Name&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="highestElevation" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Highest Elevation: The elevation from a specified vertical datum to the
highest point on a feature. [desc] In the case of multiple features that may be
stacked on each other (for example: a railway on a bridge, a superstructure on a
building, or an aerial on a tower) the highest elevation is that of the entire
feature stack. For example, the highest elevation of a church is that of its
steeple and not that of the roof of the church itself. The church itself may
have a height above surface level that excludes the additional height of the
steeple superstructure located on the church roof.'&gt;&lt;b&gt;&lt;i&gt;Highest
Elevation&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="note.memorandum" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Memorandum:   A   narrative   or   other   textual   description   that   records
observation(s)   and/or   event(s)   associated   with   a   particular   subject   (for
example: a data instance, a data set or a data processing activity). [desc] No
restriction            is            placed            on            its
length.'&gt;&lt;b&gt;&lt;i&gt;Memorandum&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayN
ame>
         </SimpleField>
         <SimpleField name="religiousInfo.religiousFacilityType" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Religious Facility Type: The type of a facility, building, structure or
site that is designed and designated to be used for religious activities, based
on   its   structure   and/or   the   principal   activity   for   which   it   was
designed.'&gt;&lt;b&gt;&lt;i&gt;Religious                              Facility
Type&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="roofShape-1" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Roof    Shape    [1]:    The    configuration(s)    and/or    appearance(s)    of    a
```

```
roof.'&gt;&lt;b&gt;&lt;i&gt;Roof                                          Shape
[1]&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="roofShape-2" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Roof   Shape   [2]:   The   configuration(s)   and/or   appearance(s)   of   a
roof.'&gt;&lt;b&gt;&lt;i&gt;Roof                                          Shape
[2]&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="specifiedEnumerants" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Specified Enumerant(s): One or more intended attribute enumerant values
for one or more enumerated attributes that are not currently valid members
of their respective attribute ranges. [desc] The actual attribute enumerant
values may have been previously, or may become in the future, valid members of
the       attribute       enumerant       range.'&gt;&lt;b&gt;&lt;i&gt;Specified
Enumerant(s)&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
         <SimpleField name="uniqueEntityIdentifier" type="string">
            <displayName>&lt;a    href='https://nsgreg.nga.mil/as/view?i=100083'
title='Unique Entity Identifier: The globally unique and persistent identifier
of an entity (for example: feature or event) instance as specified by a Uniform
Resource Name (URN) in accordance with the Internet Engineering Task Force
(IETF) RFC2396 and RFC2141. [desc] It is based on the Uniform Resource
Identifier (URI), a compact string of characters for identifying an abstract or
physical resource. The term Uniform Resource Name (URN) refers to the subset of
URI that are required to remain globally unique and persistent even when the
resource ceases to exist or becomes unavailable. The URN is drawn from one of a
set of defined namespaces, each of which has its own set name structure and
assignment            procedures.'&gt;&lt;b&gt;&lt;i&gt;Unique            Entity
Identifier&lt;/i&gt;&lt;/b&gt;&lt;/a&gt;</displayName>
         </SimpleField>
      </Schema>


      <Placemark id="BuildingGeosurface.256">
         <name>École Privée</name>
         <visibility>1</visibility>
         <styleUrl>http://portele.de/styles.kml#s1</styleUrl>
         <ExtendedData>
            <SchemaData schemaUrl="#BuildingGeosurfaceSchema">
               <SimpleData    name="type">A    free-standing    self-supporting
construction that is roofed, usually walled, and is intended for human occupancy
(for example: a place of work or recreation) and/or habitation. [desc] For
example, a dormitory, a bank, and a restaurant.</SimpleData>
               <SimpleData name="conditionOfFacility">NoInformation</SimpleData>
               <SimpleData
name="controllingAuthority">NoInformation</SimpleData>
               <SimpleData name="featureFunction-1">Education</SimpleData>
               <SimpleData name="featureFunction-2">NoInformation</SimpleData>
               <SimpleData
name="geointAssuranceMetadata.processStep.processStepDescription">No
Information</SimpleData>
               <SimpleData
name="geointAssuranceMetadata.processStep.processStepProcessor">No
Information</SimpleData>
               <SimpleData
name="geointAssuranceMetadata.processStep.processStepRationale">ETL        from
amenity_s</SimpleData>
               <SimpleData
name="geointAssuranceMetadata.processStep.source.sourceDescription">No
Information</SimpleData>
               <SimpleData name="geoNameCollection.memberGeoName.fullName">École
Privée</SimpleData>
```

```
               <SimpleData name="highestElevation">-999999</SimpleData>
               <SimpleData name="note.memorandum">No Information</SimpleData>
               <SimpleData
name="religiousInfo.religiousFacilityType">NoInformation</SimpleData>
               <SimpleData name="roofShape-1">NoInformation</SimpleData>
               <SimpleData name="roofShape-2">NoInformation</SimpleData>
               <SimpleData                         name="specifiedEnumerants">No
Information</SimpleData>
               <SimpleData name="uniqueEntityIdentifier">48681370</SimpleData>
            </SchemaData>
         </ExtendedData>
         <MultiGeometry>
            <Point>
               <coordinates>-72.69982910156267,19.110290527344</coordinates>
            </Point>
            <Polygon>
               <tessellate>1</tessellate>
               <altitudeMode>clampToGround</altitudeMode>
               <outerBoundaryIs>
                  <LinearRing>
                     <coordinates>-72.699890136719,19.110290527344             -
72.699707031250,19.110290527344 -72.699890136719,19.110290527344</coordinates>
                  </LinearRing>
               </outerBoundaryIs>
            </Polygon>
         </MultiGeometry>
      </Placemark>
      <!-- more placemarks -->
   </Document>
</kml>
```

In Google Earth, the placemark and its extended data would appear as follows:



PORT AU PRINCE HOSPITAL ASILE FRANCAIS

Building Geospatial Surface	A free-standing self-supporting construction that is roofed, usually walled, and is intended for human occupancy (for example: a place of work or recreation) and/or habitation. [desc] For example, a dormitory, a bank, and a restaurant.
Condition of Facility	NoInformation
Controlling Authority	NoInformation
Feature Function [1]	InPatientCare
Feature Function [2]	NoInformation
Process Step Description	No Information
Process Step Processor	No Information
Process Step Rationale	ETL from amenity_s
Source Description	Traced from Yahoo aerial imagery
Full Name	PORT AU PRINCE HOSPITAL ASILE FRANCAIS
Highest Elevati	[illegible]
Memor	[illegible]
Religio Facility	[illegible]
Roof S [1]	[illegible]
Roof S [2]	[illegible]
Specified Enumerant(s)	No Information
Unique Entity Identifier	48276805

Full Name: A complete name that is used to designate the entity as that designation would normally be written by the originating culture on a map or chart. [desc] It is generally considered to consist of a specific part, a generic part, and any articles or prepositions. The order of the parts may vary with the generic part appearing at the beginning, middle or end.

Annex B
(normative)

Conversion from OCL to Schematron

B.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 [5].

Naturally, the syntax representation of OCL is highly recursive. Therefore the principles of translation from OCL to another language can best be described using a recursive notation. We will 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().

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.
Free variable access, other than *self*	t 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.
Integer or real constants	123 or 3.1415		same
Boolean constants	true or false		true() or false()
String constants	‘xxxxx’		same
Enumeratio	Type::value		‘value’

Category	OCL syntax	In words	Schematron translation
n constants			
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 instance or set represented by x by applying attribute *attname*.	*If simple-typed:* τ(x)/attname *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, see B.3 *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:* τ(x.attname)[@xsi.nil='true']/@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 oclIsKindO	x . oclIsKindOf(y)	The single object instance x is	*If y is a type constant:* boolean(τ(x)[name()='T_1' or … or name()='T_i']), where

Category	OCL syntax	In words	Schematron translation
f()		checked for complying with type y.	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. Note: Expression part not implemented.
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.

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	x . concat(y)	String concatenati	concat(τ(x),τ(y)) A series of concats may be joined to a multi-

Category	OCL syntax	In words	Schematron translation
concat()		on of x and y.	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
	x implies y
		Logical combination as indicated	τ(x) and τ(y)
			τ(x) or τ(y)
			boolean(τ(x))!=boolean(τ(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	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()

Category	OCL syntax	In words	Schematron translation
		expression y(t) creates mutually different objects?	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)) 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:* 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. *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	Boolean function which yields true if the pattern of type String	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 to configure either the use of an extension function or of XPath 2.0 syntax. The XPath translation target is

Category	OCL syntax	In words	Schematron translation
	not part of the OCL standard.	matches the String argument.	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 call. See the following section B.2 for more information about the implemented ShapeChange configuration options.

B.2 Configuring the matches() operation

The OCL operation described here is currently not part oft the OCL 2.2 standard. However, discussions covering this topic found in the Internet seem to point to the expectation that the functionality is required and will probably arrive with OCL 3.0.

The expected and implemented signature of the required functionality is:

matches(pattern:String) : Boolean

matches() operates on an object *self*, which must be of type String. The result is of type Boolean and indicates whether *self* complies to the String argument *pattern* according to REGEX semantics.

There is no way to express *matches()* in XPath 1.0 except by way of using a Java extension function. XPath 2.0 indeed possesses an appropriate function. So, if the Schematron environment to be employed is capable of executing XPath 2.0 expressions, the translation process should preferably lead there.

Two configuration parameters have been added to the XmlSchema target of the ShapeChange configuration document. They control naming, namespace and arguments of the employed Java extension function, or XPath 2.0 function, respectively.

1. Configuring for use of the standard Java function matches():

```
<targetParameter
   name="schematronExtension.matches.namespace"
   value="java:java"/>
<targetParameter
   name="schematronExtension.matches.function"
   value="java.util.regex.Pattern.matches($pattern$,$object$)"/>
```

From

self.matches('\w+')

this will generate:

```
    java:java.util.regex.Pattern.matches('\w+',.)
```

Additionally a namespace definition

```
    <ns prefix="java" uri="java"/>
```

will be added to the Schematron schema document.

Though the exact way, how extension functions are to be added to XPath/XSLT environments, is not really standardized, this can be expected to work with most implementations. The namespace parameter may require specification of XPath/XSLT processor specific URLs.

2. Configuring for use of XPath 2.0:

```
<targetParameter
    name="schematronExtension.matches.namespace"
    value="fn:http://www.w3.org/2005/xpath-functions"/>
<targetParameter
    name="schematronExtension.matches.function"
    value="matches($object$,$pattern$)"/>
```

Here

self.matches('\w+')

will generate:

```
    fn:matches(.,'\w+')
```

Additionally the namespace definition

```
    <ns prefix="fn" uri="http://www.w3.org/2005/xpath-functions"/>
```

will be added to the Schematron schema document.

B.3 Configuring xlink:href reference syntax

The Schematron schema code generated from OCL invariants accompanies the ShapeChange-generated GML application schema and enables the validation of GML data beyond the syntactic validation using XML Schema.

This includes referential associations beween features, which are expressed using GML's use of simple links according to the Xlink schema. The source and target of an association, which is expressed by means of the xlink:href attribute construct has to be part of the same GML instance document.

The target point an association, which emanates from a property element carrying xlink:href is an object, which carries the gml:id attribute.

The attribute value used in the xlink:href attribute and the value in gml:id are usually not identical, because gml:id carries a label and xlink:href carries an XPointer construct, which references such a label.

The implemented translation process for xlink:href does not support XPointer. It also assumes that the source and the target of the link reside in the same document. The implementation assumes that the label contained in gml:id is somehow *contained* in the xlink:href reference in a fixed format. You can configure a prefix and a postfix, which is assumed to surround the label proper.

The defaults for prefix and postfix are as follows:

- Prefix: #
- Postfix: (empty)

This corresponds to document relative bare name syntax.

Other prefixes or postfixes may be specified by using configuration options of the XmlSchema target of the ShapeChange configuration document.

```
<targetParameter name="schematronXlinkHrefPrefix" value="xxx"/>
<targetParameter name="schematronXlinkHrefPostfix" value="yyy"/>
```

This would establish a prefix of “xxx” and a postfix of “yyy”.

B.4 Examples

This section contains examples for OCL to Schematron translations, which have been thoroughly discussed during the validation process.

B.4.1 Complex example with combined logic and forAll() iterators

The following complex OCL constraint has been manually formatted by inserting additional line breaks and indentation to achieve a better readability.

```
inv MaritimeBottomCharacterValTriples: /*Beach*/
(
  maritimeBottomCharacter.valuesOrReason.values->forAll
  ( x |
    (
        x.materialQualityOrReason.value->isEmpty()
        implies
        (
         x.materialQualityOrReason.reason->notEmpty()
         and
         x.materialQualityOrReason.reason <> VoidValueReason::valueSpecified
        )
      )
      and
      (
        x.materialQualityOrReason.value->notEmpty()
        implies
        x.materialQualityOrReason.reason = VoidValueReason::valueSpecified
      )
    )
)
and
(
   maritimeBottomCharacter.valuesOrReason.values->forAll
   ( x |
      (
        x.materialTypeOrReason.value->isEmpty()
        implies
        (
          x.materialTypeOrReason.reason->notEmpty()
          and
          x.materialTypeOrReason.reason <> VoidValueReason::valueSpecified
        )
      )
      and
      (
        x.materialTypeOrReason.value->notEmpty()
        implies
        x.materialTypeOrReason.reason = VoidValueReason::valueSpecified
      )
    )
)
and
(
   maritimeBottomCharacter.valuesOrReason.values->forAll
   ( x |
      (
        x.sedimentColourOrReason.value->isEmpty()
        implies
        (
          x.sedimentColourOrReason.reason->notEmpty()
          and
```

```
       x.sedimentColourOrReason.reason <> VoidValueReason::valueSpecified
      )
    )
    and
    (
      x.sedimentColourOrReason.value->notEmpty()
      implies
      x.sedimentColourOrReason.reason = VoidValueReason::valueSpecified
    )
  )
)
```

This constraint is translated into the following Schematron rule:

```
<rule context="nas:Beach">
  <assert test="
    not((nas:maritimeBottomCharacter/*/nas:valuesOrReason/*)[
    not(nas:materialQualityOrReason) and
    (not(nas:materialQualityOrReason[@xsi:nil='true']/@nilReason) or
    nas:materialQualityOrReason[@xsi:nil='true']/@nilReason = 'valueSpecified') or
    nas:materialQualityOrReason and
    nas:materialQualityOrReason[@xsi:nil='true']/@nilReason != 'valueSpecified']
    ) and
    not((nas:maritimeBottomCharacter/*/nas:valuesOrReason/*)[
    not(nas:materialTypeOrReason) and
    (not(nas:materialTypeOrReason[@xsi:nil='true']/@nilReason) or
    nas:materialTypeOrReason[@xsi:nil='true']/@nilReason = 'valueSpecified') or
    nas:materialTypeOrReason and
    nas:materialTypeOrReason[@xsi:nil='true']/@nilReason != 'valueSpecified']
    ) and
    not((nas:maritimeBottomCharacter/*/nas:valuesOrReason/*)[
    not(nas:sedimentColourOrReason) and
    (not(nas:sedimentColourOrReason[@xsi:nil='true']/@nilReason) or
    nas:sedimentColourOrReason[@xsi:nil='true']/@nilReason = 'valueSpecified') or
    nas:sedimentColourOrReason and
    nas:sedimentColourOrReason[@xsi:nil='true']/@nilReason != 'valueSpecified'])"
    >MaritimeBottomCharacterValTriples: Beach</assert>
</rule>
```

A short discussion to make this translation plausible:

First of all, the constraint is defined in the context of class *Beach*, which is reflected in the context attribute of the rule. The name of the constraint (the text between the introducing *inv* keyword and the colon ':') appears as the content of the assert element. The content of any comments found in the OCL constraint is appended, this can be used to create readable descriptions of assertions.

As can be easily seen, the outmost structure of the constraint is a logical *and* operation with three operands, each of which starts with a *forAll()* iterator on the property path *maritimeBottomCharacter.valuesOrReason.values.* You see this reflected in the XPath expression, which on its outmost level is also a 3-operand *and*. It will suffice to analyze one of the three operands, to understand the full expression. We will take the first one.

Now:

```
x->forAll ( t | expr(t) )
```

is not translated directly. We do it by transforming it to the equivalent expression:

```
not ( x->exists ( t | not ( expr(t) ) ) )
```

The outmost *not()* resulting from this primary step you can directly see in the result.

So, we need to negate the body of the *forAll()*.

We first replace A implies B by not(A) or B and subsequently apply De Morgan's law to achieve the negation. The result is:

```
(
 x.materialQualityOrReason.value->isEmpty()
 and
 (
  x.materialQualityOrReason.reason->isEmpty()
  or
  x.materialQualityOrReason.reason = VoidValueReason::valueSpecified
 )
)
or
(
 x.materialQualityOrReason.value->notEmpty()
 and
 x.materialQualityOrReason.reason <> VoidValueReason::valueSpecified
)
```

If you compare this to the generated XPath expression, you will immediately detect the corresponding parts, which appear in exactly the same order.

B.4.2 Translating isUnique()

This example is a shortened one, which is actually larger and uses *isUnique()* in some logic context. However, eventually this additional logic is only connected by means of an *and* and only obscures the view on the *isUnique()* subject. These parts have therefore been left away and replaced by ellipses.

```
inv VertConstMaterial: /*AircraftHangar*/
... and
verticalConstMaterial.valuesOrReason.values->isUnique(x|x)
```

This translates to the following XPath rule:

```
<rule context="nas:AircraftHangar">
 <let name="A" value="nas:verticalConstMaterial/*/nas:valuesOrReason"/>
 <assert test="...
     and
     not($A[. = (preceding::*|ancestor::*)[count(.|$A)=count($A)]])"
>VertConstMaterial: AircraftHangar</assert>
</rule>
```

The translation of *isUnique()* is a border case with XPath 1.0. A full implementation, where arbitrary expression are supported in the body of the *isUnique()* iterator, seems fully out of reach. Even the constructs supported for identity and simple attribute evaluation are quite tricky and actually rather inefficient.

Given the general form

x -> isUnique (t | y(t))

the case at hand evidently seems to be of the form *y(t)=t*, which means *identity*. *y* can also be characterized to be a collection of basic types (actually codelist values).

In the general form we translate this into the XPath expression:

not(τ(x)[.=(preceding::|ancestor::*)[count(.|τ(x))=count(τ(x))]])*

In this expression – as was defined in B.1 – *τ(x)* stands for the generated code for *x*.

x is evidently *verticalConstMaterial.valuesOrReason.values*, which means *τ(x)* (the compiled *x*) is *nas:verticalConstMaterial/*/nas:valuesOrReason*.

The latter phrase is defined on a Schematron variable *$A* by means of a let element of the rule, and you can see that the constructed assertion is exactly as planned in the general form.

Now, why does this expression assert uniqueness?

The overall construct is a *not($A[....])*, which means that the assertion is fulfilled, if *$A[...]* is the empty nodeset.

$A is the nodeset containing the collection of basic type instances which are to be proven being unique. So, the expression would be right, if the predicate rejects all elements in *$A* if and only if *$A* is unique.

The predicate compares each element (“.”) with a specially constructed nodeset, which comprises the “left” part of the InfoSet from the start of the XML document up to but not including the current element (“.” = the current basic type instance).

The “left” part of the InfoSet is *(preceding::*|ancestor::*)*, which of course also contains many elements we do not want to compare “.” to. We have to reduce the “left” part to what remains, if we intersect it with *$A*. There is no “intersect” operation in XPath 1.0, so we use the well-known count-comparison trick (you can find this in any XPath cookbook).

The trick is as follows: We compare the cardinality of *$A* with the cardinality of *$A* united (“|”) with “.”. Note that “.” here stands for the current element of the left part of the InfoSet. If “.” is not in *$A* these counts will be different.

So we end up with the intersection of *$A* with the left part of the InfoSet. And that’s what we compare to the current basic type instance. Nodeset comparisons have existence quantification, so this “=” will be true, if we find “.” in the “left part of *$A*”. Which means that “.” is identical to another instance in *$A* and which is exactly the opposite of uniqueness.

Please note that, though this construct works, it can be supposed to be quite slow. Reason: Any value is compared to the left part of the tree, which is of quadratic order. And additionally we have the trick providing the intersection operation, which is also quadratic. This further increases the order to $O(n^4)$.

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

[4] OGC® Web Feature Service, version 1.1.0, OGC document 04-094 (OGC standard)